'68' MICRO JOURNAL

$2.50 USA

Australia	A $ 4.00	New Zealand	NZ $ 4.00
Singapore	S $ 8.00	Hong Kong	H $20.00
Malaysia	M $ 8.00		

VOLUME II ISSUE 4 • Devoted to the 68XX User • April 1980

"Small Computers Doing Big Things."

SERVING THE 6800 USERS WORLDWIDE

PHOTO CREDIT: NASA

SYSTEMS - SOLUTIONS

If you have a problem that can be solved by a computer—we have a systems solution.

- Two central processors with maximum RAM capacities of 56K and 384 K bytes
- Three types of disk drives with capacities of 175K, 1.2M and 16M bytes
- Two dot matrix printers with 80 and 132 line capacity
- A Selectric typewriter interface and a daisy wheel printer

Match these to your exact need, add one or more of our intelligent terminals and put together a system from one source with guaranteed compatibility in both software and hardware.

Southwest Technical Products systems give you unmatched power, speed and versatility. They are packaged in custom designed woodgrain finished cabinets. Factory service and support on the entire system and local service is available in many cities.

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

The Source for 6800/6809 Software

Technical Systems Consultants, Inc. is The Source for your 6800/6809 systems software needs. From FLEX™, the standard disk operating system of the 680X family, to Sort/Merge, your systems requirements can be filled with the highest quality software in the industry. Nowhere else can you find such variety from a single source. Here are some of the most popular:

Program Name	6800	6809
FLEX for SWTPc	$90	$90
FLEX for SSB	90	90
Extended BASIC	100	100
Extended BASIC Precompiler	50	50
BASIC	65	65
BASIC Precompiler	40	40
FLEX Sort/Merge	75	75
Text Editing System	40	35
Assembler	40	40
Text Processing System	60	N/A
Debug Package	55	75
FLEX Utilities	100	60

These packages are available on either 8" or 5" soft-sectored FLEX diskettes (5" 6800 is FLEX 2.0). Price includes user's manual and object code diskette. Certain programs are available on cassette. Contact Technical Systems Consultants for pricing. All orders should include 3 percent for postage and handling (8 percent on foreign orders). Master Charge and Visa are welcome.

™FLEX is a trademark of Technical Systems Consultants, Inc.

technical systems consultants, inc.
Box 2574, West Lafayette, IN 47906
(317) 463-2502

'68' MICRO JOURNAL

Portions of text prepared using the following.

SWTPC 6800-6809-DMAF2-CDS1-CT82-Sprint 3
Southwest Technical Products
219 W. Rhapsody
San Antonio, Texas 78216

GIMIX Super Mainframe-Assorted memory boards
GIMIX Inc.
1337 West 37th Place
Chicago, Il 60609

Publisher: Don Williams Sr.

Executive Editor: Larry Williams

Subscriptions and Office manager
Mary Robertson

General Girl 'Friday'
Joyce Williams

Contributing Editors:

Mickey Ferguson
Dennis Womack
Dr. Jack Bryant
Dr. Chuck Adams
Dr. Theo Elbert
Dr. Jeffery Brownstein
Dale Puckett
Russell Gore
T. Jackson, Japan

Typography and color work:
Williams Inc.
Chattanooga, TN 37421

* CONTENTS *

Send All Correspondence To:

'68' Micro Journal
3018 Hamill Rd.
PO Box 849
Hixson, Tennessee 37343

— Phone —
Office: 615-870-1993
Plant: 615-892-7544

'68' Micro Journal is published 12 times a year by '68' Micro Journal, 6131 Airways Blvd., Chattanooga, TN 37421. Second Class postage paid at Chattanooga, TN. Postmaster: Send Form 3579 to '68' Micro Journal, PO Box 849, Hixson, TN 37343.

1-Year $14.50 2 Years $26.00 3 Years $36.50

—ITEMS SUBMITTED FOR PUBLICATION—

(Letters to the Editor for Publication) All 'letters to the Editor' should be substantiated by facts. Opinions should be indicated as such. All letters must be signed. We are interested in receiving letters that will benefit or alert our readers. Praise as well as gripes is always good subject matter. Your name may be withheld upon request. If you have had a good experience with a 6800 vendor please put it in a letter. If the experience was bad put that in a letter also. Remember, if you tell us who they are then it is only fair that your name 'not' be withheld. This means that all letters published, of a critical nature, cannot have a name withheld. We will attempt to publish 'verbatim' letters that are composed using 'good taste.' We reserve the right to define (for '68' Micro) what constitutes 'good taste.'

(Articles and items submitted for publication) Please, always include your full name, address, and telephone number. Date and number all sheets. TYPE them if you can, poorly handwritten copy is sometimes the difference between go, no-go. All items should be on 8X11 inch, white paper. Most all art work will be reproduced photographically, this includes all listings, diagrams and other non-text material. All typewritten copy should be done with a NEW RIBBON. All hand drawn art should be black on white paper. Please no hand written code items over 50 bytes. Neatly typed copy will be directly reproduced. Column width should be 3¼ inches.

(Advertising) Any Classified: Maximum 20 words. All single letters and/or numbers will be considered one (1) word. No Commercial or Business Type Classified advertising. Classified ads will be published in our standard format. Classified ads $7.50 one time run, paid in advance.

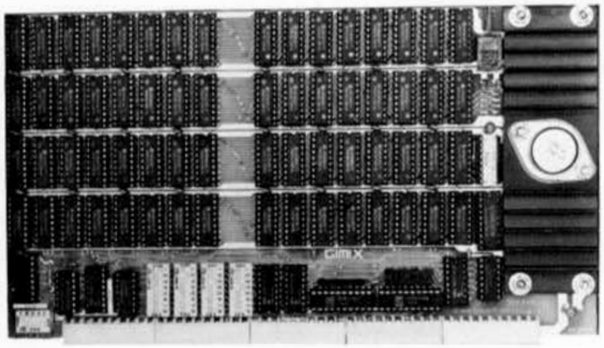
Commercial and/or Business advertisers please write or phone for current rate sheet and publication lag time.

WANT TO BE KING OF THE HILL?

TREAT YOURSELF ROYALLY WITH GIMIX UNIQUE AND INCOMPARABLE BOARDS AND SYSTEMS . . . DIP-switch Versatility for use with both SS50 (6800) and SS50C (6809) Systems (SWTP. etc.)

32K STATIC RAM BOARDS

Designed for use with: ★ Existing SS50 Systems
★ SS50C Extended Address Systems



- Assembled
- Burned In
- Tested

16K	$328.12
24K	$438.14
32K	$548.15

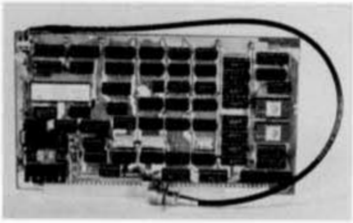
16K and 24K Versions are socketed for 32K and require only additional 2114's for expansion.

FEATURES:

- Decoding for 4 Extended Address Lines (allows memory decoding up to 1 megabyte)
- DIP-switch to set extended addressing or disable it
- 4 separate 8K blocks, addressable to any 8K boundary by DIP-switch
- Each 8K block may be individually disabled
- Write protect either of two 16K sections
- Low power consumption — uses 2114L low power RAMS — (2 amps typical for 32K)
- Fully Socketed
- Gold Bus Connectors
- Guaranteed 2MHz operation

FACTORY PRIME STATIC RAMS				
	2114L 450 ns	$5.90	200 ns	$6.90
	4044 450 ns	$5.90	250 ns	$6.90

ADD $5.00 HANDLING ON ORDERS UNDER $200.00



THE UNIQUE GIMIX 80 x 24 VIDEO BOARD

★ Upper and Lower Case with Des enders
★ Contiguous 8x10 Character Cells
★ Hardware Scrolling
★ X-Y Addressable Hardware Cursor

It is the ***ONLY*** **Video Board** that gives you **Software Control** of:

- A programmable RAM Character Generator plus 2 EPROM Character Generators (128 char. ea.).
- Selecting 256 Displayable Characters from 384 available.
- Normal or Inverse video, full or reduced intensity, or combinalions of these by *both* ASCII Code and Bit 8.
- GHOSTability — multiple boards at the same address.

Fully decoded, occupies only 2K of address space
Fully socketed — Gold bus connectors.
Assembled, Burned-in, and Tested at 2MHz.

Deluxe Version	$458.76
Without RAM Character Generator .	$398.74
Other Video Boards from	$198.71

I.O. BOARDS

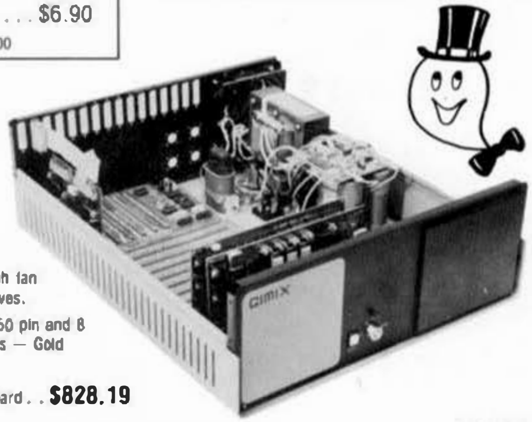
NEW!! 50 PIN 8 PORT RS-232 SERIAL BOARD

★ With SS50C extended addressing (can be disabled)
★ DIP-switch selectable Baud Rates for each Port
★ Occupies 16 bytes of address space
★ Dip switch addressable to any 16 byte boundary
★ Any 2 of IRQ, NMI or FIRQ can be used
★ RS-232 handshaking for each port (CTS, RTS, DCD) **$288.40**
★ With optional on Board Baud Rate Generator **$318.46**

ALSO AVAILABLE

50 Pin 8 Port Parallel	$198.45
30 Pin 2 Port Parallel	$ 88.42
30 Pin 1 Port Serial (RS 232 or 20MA. current loop)	$ 88.41
Cable Sets for above . . . . ea.	$ 18.95

THE CLASSY CHASSIS



- Ferro-resonant Power Supply
- Heavyweight Aluminum Cabinet with fan and provisions for two 5" disk drives.
- 6800/6809 Mother Board, fifteen 50 pin and 8 DIP-switch addressable 30 pin slots — Gold Plated Pins. Fully decoded.
- Baud Rate Generator on Mother Board . . **$828.19**

32K SYSTEM *Incomparable Features at a Comparable Price!* **$1,594.59**
Includes: Chassis, 6800 CPU, 32K RAM Board, Choice of I/O Card

16K Version of above **$1,374.49**

EXPORT NOTES:

For 50Hz 230V C.V. POWER SUPPLY . . . . Add $30.00
80 x 24 VIDEO BOARDS — Specify Format (No Added Charge)

On Orders under $250.00 for a Single Board, or Chips, please Add $30.00 Handling and we will ship Air Mail Prepaid.

On all other orders we will ship via Emery Air Frieght Collect, and we will charge no handling. All orders must be prepaid in U.S. Funds. Please note that foreign checks have been taking about eight weeks for collection, so we would advise wiring money or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, account #73-32033. Visa or Master Charge also accepted.

GIMIX® and GHOST® are Registered Trademarks of GIMIX INC.

Phone, write, or see your dealer for details and prices on our broad range of Boards and Systems for the SS50/SS50C bus and our AC Power Control Products for all computers.

GIMIX inc.

The Company that delivers
Quality Electronic products since 1975.

1137 WEST 37th PLACE, CHICAGO, IL 60609
(312) 927-5510 • TWX 910-221-4055

SEE GHOST AD PAGES 39 & 44

M6800 SPL/M COMPILER

PROGRAMMA

SPL/M is a block-structured language which features arbitrary length identifiers and structured programming contructs. It is suitable for systems programming on small computers, since the compiler requires only 20K of memory and one disk unit to run.

The language can be compiled in only one pass, which means that the source code has to be read only once.

Unlike most high-level language translators available for micro processors, SPL/M is a true compiler: it generates absolute 6800 object code which requires no run-time package. Due to extensive intra-statement optimization, the generated code is almost as efficient as the equivalent assembly language.

The compiler has a number of compile-time options, including a printout that contains the interlisted object code. Syntactical error messages use position indicators to indicate exactly where an error occurs.

SPL/M is currently available for the SWTPc 6800 (or other compatible systems) running under the FLEX 2.0 or the SSB DOS operating system. The system is supplied on one 5¼" diskette together with a comprehensive User's Manual.

Statement Types

DECLARE
Assignment (e.g. COUNT = 1;)
IF-THEN (with optional ELSE)
DO-WHILE (loop control, plus BREAK)
Grouping (DO; statement list END;)
PROCEDURE definition
GENERATE machine code

Declarations

Define variable types, either BYTE (8 bits),
    or ADDRESS (16 bits)
Define arrays (one dimension only), either
    variable or constant (DATA)
Define compile time numeric substitutions
    (LITERALLY)

Operators

Arithmetic:
    + - * / MOD (MODULO)
Logical:
    NOT AND OR XOR
Relational:
    = = (not equal)

Procedures

Defined within a PROCEDURE-END pair and
    called via a CALL statement

Functions

Type conversion (LOW, HIGH, DOUBLE)
Direct memory access (MEM, MEMA)

Miscellaneous

Identifiers may be up to 31 characters long
Decimal, hexadecimal, and string constants
Integer arithmetic only
Origins for program and/or oata

PROGRAMMA
International Inc.
3400 Wilshire Blvd.
Los Angeles, CA 90010

(213) 384-0579 384-1116 384-1117

$49.95

plus postage

Software Products

SMOKE SIGNAL BROADCASTING

Presents

3 Powerful New SS-50/SS-50C Boards

DCB-4
Disk Master
Double Density Controller Board and DOS68D Double Density DOS
$449.00

The new DCB-4 is a truly state-of-the-art development which allows up to 366K bytes to be stored on a single 5¼" disk and has these outstanding features:

- Up to four 5¼"and four 8" drives can be handled in the same system with a user definable logical unit table. (DOS68D will be compatible with future hard disk systems).
- Under software control, the user can select the following for any drive:
  - ✩ Single sided or double sided operation.
  - ✩ Single density or double density data.
  - ✩ 5¼" or 8".
  - ✩ Stepping Rate.
  - ✩ 40 track or 35 track density on double sided 5¼" drives.
  - ✩ User can select the system boot configuration.
- Occupies only 16 bytes of memory space (F760-F76F standard). User selectable to any 16 byte address space.
- Can read and write a single sector by itself. On-board buffer memory allows full interrupt capability in interrupt driven systems. Once data transfer has been initiated, no more processor time is required.
- Contains extended decoding circuitry for extended addressing per SS—50C bus which can be enabled by an option jumper.
- SSB provides a means for copying software written by older versions of DOS68 to be read by DOS68D. All new media formatted by DOS68D can be read by all older versions of DOS68. DOS68 is SSB's 6800 disk operating system.
- Track Ø of side Ø is recorded in single density per IBM standard.
- Phase-locked-loop assures highest data integrity attainable.

All of these features are available for immediate delivery on one standard 5½" x 9" 50 pin SS—50/SS—50C card for only $449.00. The price includes DOS68D version 5.1, MONITOR object code on diskette, and a manual with the source listing.

SCB-69
Super Computer Board
6809CPU Board
$299.00

The most versatile 6809 CPU Board on the market is now available from Smoke Signal Broadcasting and has the following features:

- Standard 2 MHz operation. (Shipping 1.5 MHz until August 80)
- 20 bit address generation for up to 1 Mbyte of memory. Uses an improved address translation RAM which is compatible with present extended addressing schemes yet requires much less overhead when used in multi-user systems.
- All on-board devices can be switch selected to occupy any or all extended pages. Any on-board device may be disabled and its memory space is then available for external memory.
- Standard real—time clock (time—of—day, day—of—week, day—of—month) with battery back up capable of generating programmable interrupts.
- Up to 20K of EPROM can be installed on the CPU Board.
- Standard 1K of RAM on board.
- Includes improved 6809 Monitor (and source listing).
- Contains an FPLA for decoding EPROM address and optional devices. Switches are used to select 2K/4K EPROM and Fast/Slow I/O.
- Contains provision for optional 9511/9512 floating point processor.
- NMI line is user selectable to work with either SS—50 or SS—50C busses.

Price for the new SCB—69 is only $299.00 for an assembled, burned-in fully tested board.

M-32-X
32K
Memory Board
$539.00 $439.00

The first and only 32K Static Ram Board on standard size (5½" x 9") SS—50/SS—50C Bus Circuit Card is made by Smoke Signal.

- Switch selectable to any 4K boundary.
- Any 4K block may be switch enabled or disabled.
- Fully compatible with SS—50C extended addressing (allows memory decoding up to 1 Mbyte).
- Extended addressing capability may be switched off for compatibility with SS—50 systems.
- Gold Bus Connectors for high reliability.
- Guaranteed 2MHz operation (tested at 2.2 MHz).
- Low power consumption — 8 volts at 2.4 amps typical.

M—32—X 32K Memory Board is priced at $539.00.

M—24—X 24K Memory Board expandable to 32K, is $439.00.

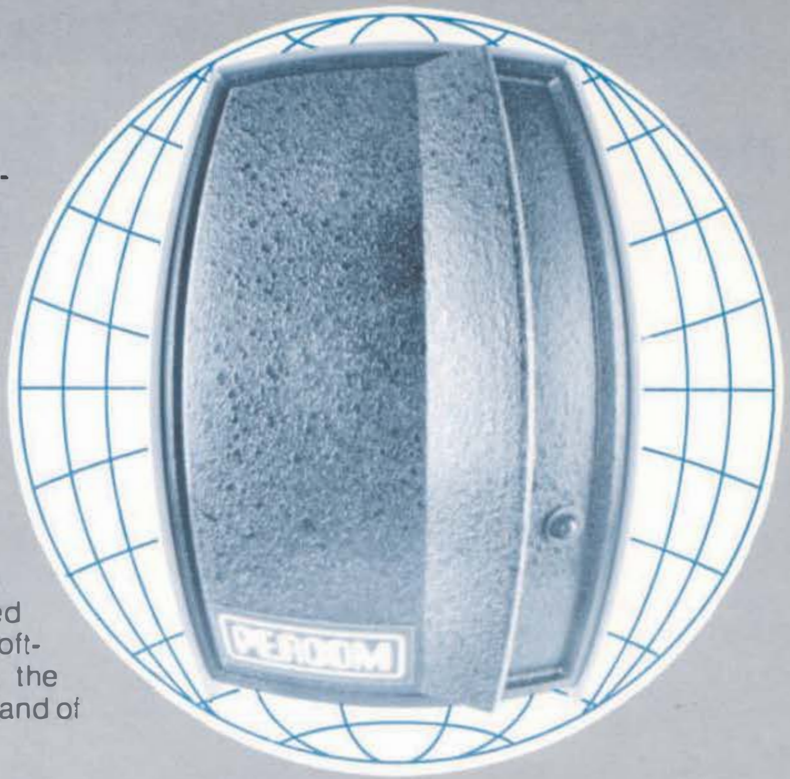
And our M—16—X 16K board is back to the old price of $299.00.

SMOKE SIGNAL BROADCASTING®

31336 Via Colinas, Westlake Village, CA 91361, (213) 889-9340

In the world of 6800 Microcomputing there is only one Universal Mini-Disk System . . .

the PERCOM LFD-400 ™ with SOFTRAN ™



Made possible by SOFTRAN™, an innovative $24.95 translator program, the reliable Percom LFD-400™ has just been upgraded to the first universal mini-disk storage system.

Suddenly the two worlds of 6800 minidiskette software become one. Because the LFD-400™ with SOFTRAN™ can read either soft-sectored or hard-sectored disks.

And owning an LFD-400/SOFTRAN system means you can run minidiskette programs from the enormous combined selection of all of the principal 6800 software houses — TSC, Computerware, the Software Works, Hemenway Associates and of course Percom.

Available in versions for mini FLEX†, FLEX 2.0† and Smoke Signal Broadcasting Company's DOS, SOFTRAN™ copies soft-sectored minidiskettes track-for-track onto hard-sectored minidiskettes. If the source disk includes a FLEX† or 'Smoke' DOS, SOFTRAN™ is used to modify the operating system to function with the Percom LFD-400™.

SOFTRAN™ is supplied on a minidiskette along with utilities for only $24.95. A users manual is included. You must indicate whether SOFTRAN™ is to be used for mini FLEX†, FLEX 2.0† or Smoke's DOS.

The Percom LFD-400™ mini-disk system sells for only $599.95, complete with: (1) the drive, drive electronics and Percom's rugged PS-401 power supply all in a finished enclosure, (2) a demonstrably superior controller PC card featuring an explicit data/ clock separation circuit, MPX, a remarkable 2K DOS, and provision for 1K extra PROM, (3) an interconnecting cable and (4) a 70-page users manual.

Also available: Upgrade kits for SWTP or 'Smoke' mini-disk drive systems. Kit includes LFD-400™ controller, MPX DOS & SOFTRAN™. Only $224.95.

Available soon!
SOFTRAN™ for Percom's 77-track LFD-800™ mini-disk system; SOFTRAN/9™ for 6809 FLEX† files and programs.

™ trademark of Percom Data Company, Inc.

† trademark of Technical Systems Consultants, Inc.

PERCOM

PERCOM DATA COMPANY, INC.
211 N. KIRBY GARLAND, TEXAS 75042
(214) 272-3421

Percom *'peripherals for personal computing'*

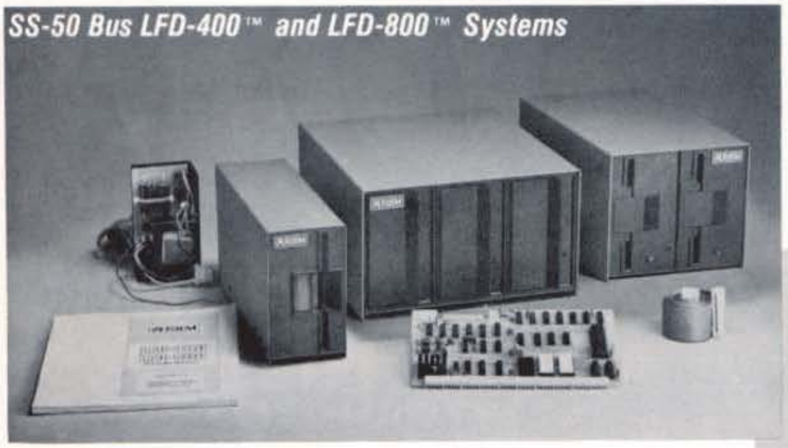
To place an order or request additional literature call toll-free 1-800-527-1592. For technical information call (214) 272-3421. Orders may be paid by check, money order, COD or charged to a VISA or Master Charge account. Texas residents must add 5% sales tax.

PRICES AND SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE

A Few Extraordinary Products for Your 6800/6809 Computer

From Percom . . .

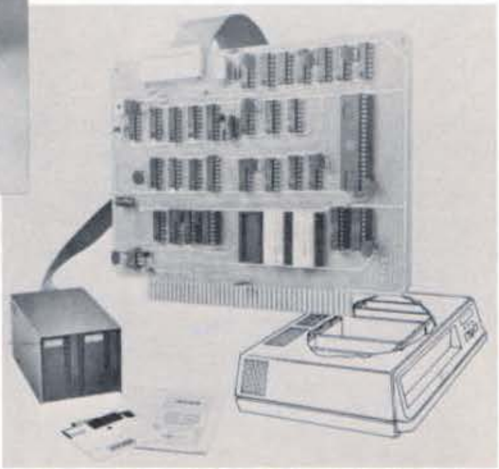
Low Cost Mini-Disk Storage in the Size You Want



SS-50 Bus LFD-400™ and LFD-800™ Systems

Percom mini-disk systems start as low as $599.95, ready to plug in and run. You can't get better quality or a broader selection of disk software from any other microcomputer disk system manufacturer — at any price!

Features: 1-, 2- and 3-drive systems in 40- and 77-track versions store 102K- to 591K-bytes of random access data on-line • controllers include explicit clock/data separation circuit, motor inactivity time-out circuit, buffered control lines and other mature design concepts • ROM DOS included with SS-50 bus version — optional DOSs for EXORciser* bus • extra PROM sockets on-board • EXORciser* bus version has 1K-byte RAM • supported by extended disk operating systems; assemblers and other program development/debugging aids; BASIC, FORTRAN, Pascal and SPL/M languages; and, business application programs.



EXORciser Bus LFD-400EX™ -800EX™ Systems*

Full Feature Prototyping PC Boards

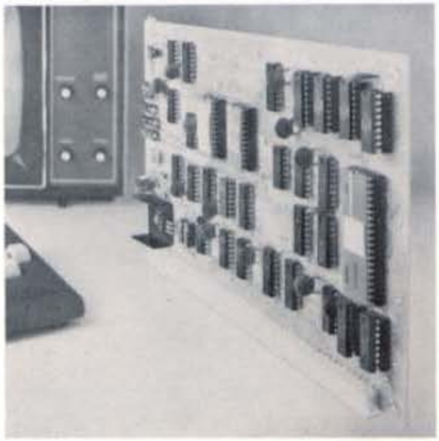
Percom SS-50 and I/O bus prototyping cards include all of the features needed for easy, straightforward prototyping. Use wire wrap, wiring pencil or solder wiring. Features: tin-lead plating over 2 oz. copper wets quickly, solders easily • provision for power regulators and distributed capacitor bypassing • SS-50 bus card accommodates 34- and 50-pin ribbon connectors on top edge, 10-pin Molex connector on side edge — costs only $24.95 • I/O bus card is 1-1/4" higher than SWTP I/O card, accommodates 34-pin ribbon connector and 12-pin Molex connector on top edge — costs only $14.95 • Both card designs accept 14-, 16-, 24- and 40-pin DIP sockets.

Upgrade to 6809 Computing Power

This 6809 upgrade adapter may be used on the SWTP 6800 and most other 6800/6802 MPU cards. Supplied assembled and tested, it costs only $69.95 with user instructions. The original system may be restored by merely unplugging the adapter and a wire-jumpered DIP header, and re-inserting the original components. Also available for your upgrade computer is PSYMON™, the Percom SYstem MONitor for the Parcom 6809 single-board computer. PSYMON™ on 2716 ROM costs only $69.95 — PSYMON™ is also available on minidiskette, with source and object files, from the Percom Users Group.

The Electric Window™: Instant, Real-Time Video Display Control

This VDC card resides completely in main memory so that control is accomplished instantly by direct MPU access to the on-board 2K character-store memory and the display control registers. Price is only $249.95. **Features**: Programmable CRT controller chip provides extraordinary capability for software control of functions such as number of characters per line, number of lines displayed, highlighting and interlaced or non-interlaced scan • includes ASCII 128-unit character generator which generates 7-dot by 12-dot characters — lower case letters have descenders • provision for optional ROM for special characters/symbols • comprehensive manual includes full listing of WINDEX™, the Eteclric Window™ driver program — WINDEX™ is also available on minidiskette through the Percom Users Group.



™ trademark of Percom Data Company, Inc.
* trademark of the Motorola Corporation
Prices and specifications subject to change without notice.

PERCOM

PERCOM DATA COMPANY, INC.
211 N. KIRBY GARLAND, TEXAS 75042
(214) 272-3421

Products are available at Percom dealers nationwide. Call toll-free, 1-800-527-1592, for the address of your nearest dealer, or to order direct.

SPLM-A Systems Language

Dale Puckett
14753 Endsley
Woodbridge, VA 22193

Programmers that have been SEARCHing for a language that allows them to talk to their machines at the gut level and still use long, highly understandable names for labels and variables, can set the FOUND flag. SPL/M is here.

SPL/M stands for Small Programming Language for Microprocessors and is based on PL/M which was developed by Intel. It is a block structured language and allows the names of variables and procedures to be up to 31 characters long. It was written by Thomas W. Crosley and Programma, International, Inc. and is sold for $59.95. It is available for FLEX 1.0 or FLEX 2.0, but not miniFLEX. Programma's address is 3400 Wilshire Boulevard, Los Angeles, Ca., 90010.

SPL/M is small and requires only 20K to run although a disk or two cassette recorders are required. It compiles your program in only one pass and generates absolute 6800 object code. No run-time package is required.

This review will list and explain the major SPL/M reserved words and structures. A sample program called WHERESIT, designed to illustrate the simplicity of the language's structure and readability while detailing a very useful utility for all FLLEX users, is included.

The reserved words in version one of SPL/M include:

PROCEDURAL	MEMORY RELATED	LOGICAL
BREAK	ADDRESS (ADDR)	AND
CALL	BYTE	NOT
DECLARE (DCL)	DATA	OR
DO	HIGH	XOR
ELSE	LITERAALLY (LIT)	
END	LOW	
EOF	MEM	
GENERATE (GEN)	MEMA	
IF	. (returns an absolute address)	
PROCEDURE (PROC)		
RETURN		
THEN		
WHILE		

The words in parenthese are legal substitutions for the full words they follow.

Programmers may use any identifiers, except the reserved words, up to 31 characters long. The first letter must be alphabetic. The rest may be letters, numbers, or the separation character, "$". In the program WHERESIT, for example, VERSION is the first identifier. GET$INFO$RECORD is also an identifier — one which leaves little doubt to its purpose.

ASSIGNMENT

In an SPL/M program all identifiers must be declared before they are used. To do this you either use the DECLARE verb to name all variables and symbolic constants and the PROCEDURE verb to define procedures.

Constants are either unsigned numbers, within the range of 0 to 65535 or character strings. They are initialized with a DECLARE statement. In the sample program the statement, DECLARE WORKDRIVE LITERALLY '1', assigns the value 1 to the constant WORKDRIVE.

Memory locations for variables are reserved in the same manner. For example the line, DECLARE STOPCUE BYTE, reserves one byte for the variable, STOPCUE.

Additionally the line, DECLARE DISKNUMBER ADDRESS, reserves two bytes for the variable DISKNUMBER. A one dimensional array or vector may also be reserved. For example, DECLARE RFC8(320) BYTE, reserves 320 bytes for a file control block. After it is declared, any position in the array may be referenced. For instance, RFCB(4) would normally contain the first character of a FLEX filename. By the way, these declares can be directed to a specific address. Suppose you want SPL/M to reference FLEX's width parameter byte as a program variable called WIDTH. You would simply type, 0AC04H: DECLARE WIDTH BYTE.

SPL/M's operators are standard and easy to learn, for example BUFSIZE = BUFSIZE - 1, subtracts one from the value of BUFSIZE. BUF5 = BUF5 + 1, adds one to the contents of the variable BUF5.

Two special SPL/M verbs, similar to PEEK and POKE functions in BASIC, are MEM and MEMA. They allow a direct reference to memory. MEM handles a single BYTE variable, MEMA handles an ADDRESS or 16-bit word.

In WHERESIT, the procedure WRITE$IT illustrates the use of MEM. The statement CHAR = MEM(BUF5) sets CHAR to the value of the byte pointed to by BUF5. In this case it is similar to the BASIC PEEK function.

In the procedure TRANSFER$DISK$NAME, the statement MEM(BUF2) = MEM(BUF1) sets the value of the byte pointed to by BUF2 to the value of the byte pointed to by BUF1. In other words, when it is located on the left side of the assignment statement, MEM acts like the BASIC POKE function.

OBJECT CODE INTERFACE

The statement GENERATE or its short version GEN allows the programmer to talk to memory directly. Here is a sample statement which uses this verb.

GEN(ODEH, .MSGA).

This line will load the 6800's X-register with the address of the variable MSGA. MSGA is assumed to be on the first page in this case since direct addressing is implied by the "DE."

This sample is actualy a major part of a library function already supplied called PSTRNG. When it is called the function simply points the X-register to the character pointed to by MSGA and then calls FLEX's PSTRNG routine. Here's a very good point. All of the basic FLEX DOS routines are already interfaced and the SPL/M user doesn't have to reinvent the wheel. He only has to INCLUDE the proper library at the front of his program.

Three library files are located on the disk you receive from Programma. They are named SPLM.LIB, an interface between the compiler and the FLEX DOS routines; SPLMREAD.LIB, a collection of routines necessary to read a sequential file; and SPLMWRIT.LIB, which contains the code necessary to write sequential files.

STRUCTURE AND FLOW

SPL/M does not have a GOTO statement. Instead it controls the flow of programs through the use of three common programming constructs, IF-THEN-ELSE, DO-END, and DO-WHILE, There are actually two forms of the IF-THEN construct and the first is used in the WHERESIT procedure DELETE$TEST. IF (CHAR AND DELETE$MASK) THEN DELETE = TRUE will set DELETE = TRUE if there is a character in CHAR and IF the eighth bit of the CHAR is set. If either of these conditions is false then the routine will fall through to the end statement and return.

The other IF-THEN construct is used in $GET$INFO$RECORD to clarify which IF owns the ELSE. Oh well, a picture is worth a thousand words.

```
IF ERROR THEN DO;
                 IF RFCB(XES) = EEOF THEN RETURN;
                 ELSE CALL HANDLE$ERRORS;
                 END;
     REOF = FALSE;
END;
```

A close observer will notice that a second construct, DO-END was also used in this procedure. It was necessary to force the ELSE to belong to the second IF statement. Since the ELSE was inside the DO-END limits, it is isolated from the first IF statement.

Finally, the third construct is illustrated in the procedure TRANSFER$DIRECTORY. The statement, DO WHILE NOT REOF does just what it says. It forces the program to loop until REOF becomes true and then it exits.

Another handy feature of SPL/M is the ability to call a machine language procedure by its address. For example you might want to call SWATBUG's clear screen routine at $E2CC. To do this with SPL/M you may simply CALL 0E2CCH and the proper code will be compiled. Of course your program will be more understandable if you DECLARE CLEAR$SCREEN LIERALLY '0E2CCH' and then use the statement CALL CLEAR$SCREEN when you want to use the function.

OPERATION

Your program can be given an absolute start address by using an optional orgin statement, ie, 0A100H:; or it may be allowed to default to 0100 hex. Procedures may also be placed at specified locations by using an orgin statement.

If you are sold on the simplicity and beauty of SPL/M code by now, you probably want to know how it operatas.

On the disk from Programma you will find the following files: SPLM.CMD, the actual compiler; FLXI02.TXT, the assembler source code for the compiler's interface to FLEX; SPLM.LIB, SPLMREAD.LIB, and SPLMWRIT.LIB, described above; and SIZE.TXT, a sample program written in SPL/M source code so you may test your new compiler. SIZE is a very useful utility which raports the date a file was created, the number of sectors it contains, the number of bytes in the file, in both decimal and hexidecimal, the number of lines in the file and a checksum.

To compile a source program which you have written to a text file you simply type: SPLM, SOURCENAME, BINARYNAME, +OPTION LIST.

The BINARYNAME and OPTION LIST are optional and if you type only SPLM SOURCENAME your binary file will be given the filename, SOURCENAME.BIN. If you type only SPLM (CR), the compiler will operate in an interactive mode and the code you are generating is printed on the terminal. This is an extremely educational experience. You get to see how the different constructs work at the machine level, not to mention the fact that you can pre-test short procedures and see how they work.

The OPTION LIST can be any of the following letters: B Y E C G A. The letters may be typed in any order.

The B allows you to prevent the creation on an object code file. The Y lets you automatically delete an old object file with the same name, the E tells the compiler to display the errors only, the C causes the compiler to output a full listing which includes the object code, G causes it to output a symbol table with all global symbols, and A causes it to output a symbol table which includes all symbols, both local and global.

Finally, to obtain a listing on your printer you only need to use the FLEX P.CMD utility, ie, P SPLM SIZE.

CONCLUSION

SPL/M is a very powerful language, especially tor systems level programs. Its programs are easy to write. And, most importantly, they are very easy to read. If you've looked at a program you wrote in BASIC six months ago, you can't help but know how important readability is. With SPL/M's unbelievable low price and power it has to be one of the best buys in the 6800 software field today.

EDITOR'S NOTE: For '68' Micro Journal readers interested in running WHATSIT that do not have the SPL/M compiler, Dale will sell a copy of the object code plus a source listing with symbol table on a disk for $15. If you send him a disk and a self-addressed, stamped envelope, he will send it to you for $10.

A 68 Micro Journal™ lab rating of AAA.

Rating Scale:
AAA - Excellent
AA - Good
A - Fair (could be better but works)
P - Poor (may not always work properly)
X - Not recommended for children
(or anything else!)

WHERESIT

WHERESIT.CMD outputs a mini-directory which contains the filename, extension, date created, disk name, and disk number.of every disk inserted. Its output looks like this:

```
LOSTFILE CMD 12-17-79 ASMBWORK 1022 (CR)
```

The inclusion of the disk name and disk number on each line allows you to find a program which you wrote six months ago and misplaced. It is especially useful when you have more than four or five disks and saves the time it takes to CAT each disk when looking for a misplaced file.
To build an index file containing a directory of all your disks:
Place the disk which will contain your index file in drive 0. Place a copy of WHERESIT on the same disk. Then, plece the first disk to be filed into drive 1 and type:

```
+++0, 0.ALLDISKS, WHERESIT
```

After you have read the directories from all of your disks type an "E" in response to the prompt.
Now, you may use the FIND.CMD to locate a misplaced file. Type:

```
+++FIND, ALLDISKS, LOSTFILE
```

Each line in ALLDISKS which contains LOSTFILE will be printed on the terminal. That line will give you the number and name of the disk containing your program.

If desired, ALLDISKS may be sorted by filename, extension, or date created, etc., with the SORT/MERGE package. The results could then be printed into an alphabetical index to every file you own. Or, use your imagination.

ALLDISKS may be updated by running WHERESIT once a month or whenever you get the urge. It only takes about three minutes to build an index of 25 disks using WHERESIT.

Since SPL/M is structured so nicely the user could easily modify this source code to read other information from his disk directories. */

```
0A100H:;  /* Starting address */
/* Variables and procedures not declared or defined
   below are defined in SPLM.LIB, a library file which
   is read by this program. */

DECLARE VERSION LITERALLY '1';
0100H: DECLARE RFCB(320) BYTE;
DECLARE DELETE$MASK LITERALLY '80H';
DECLARE DELETE BYTE;
DECLARE REOF BYTE;
DECLARE STOPCUE BYTE;
DECLARE ENDFLAG BYTE;
ONCLUDE SPLM.LIB
DECLARE WORKDRIVE LITERALLY '1';
DECLARE QOS14R LITERALLY '16';
DECLARE QOD4R LITERALLY '6';
DECLARE QRDIR LITERALLY '7';
DECLARE DISKNAME(8) BYTE;
DECLARE DISKNUMBER ADDRESS;
DECLARE BUF1 ADDRESS, BUF2 ADDRESS, BUF3 ADDRESS, BUF4 ADDRESS;
DECLARE BUF5 ADDRESS;
DECLARE BUFSIZE BYTE;

WRITE$IT:PROCEDURE;                   /* Outputs the characters */
    DO WHILE BUFSIZE <> 0;             /* pointed to by BUF5.  String */
       CHAR = MEM(BUF5);              /* is BUFSIZE characters long. */
       IF CHAR = 0 THEN CHAR = ' ';      /* If a character is null */
       CALL PUTCHR;                    /*  a space is printed. */
       BUF5 = BUF5 + 1;
       BUFSIZE = BUFSIZE - 1;
    END;
END;

/* All disk errors except end of file are treated
   as fatal. */

HANDLE$ERRORS:PROCEDURE;
      FCBA = .RFCB;
      CALL RPTERR;
      CALL WARMS;
END;

/* Point to address containing the disk number and
   print it. */

WRITE$DISK$NUMBER:PROCEDURE;
      LDSPC = TRUE;
      DGTA = .DISKNUMBER;
      CALL OUTDEC;
END;

/* Point to buffer containing the disk name and
   print it. */

WRITE$DISK$NAME:PROCEDURE;
      BUF5 = .DISKNAME;
      BUFSIZE = 8;
      CALL WRITE$IT;
END;

/* Point to extension in RFCB and print */

WRITE$EXTENSION:PROCEDURE;
      BUF5 = (.RFCB + 12);
      BUFSIZE = 3;
      CALL WRITE$IT;
END;

/* Point to filename in RFCB and print it. */

WRITE$FILENAME:PROCEDURE;
      BUF5 = (.RFCB + 4);
      BUFSIZE = 8;
      CALL WRITE$IT;
END;

/* Check bit eight of first character of filename.
   If set this file has been deleted. */

DELETE$TEST:PROCEDURE;
      DELETE = FALSE;
      CHAR = RFCB(XFN);  /* GET FIRST CHAR OF FILENAME */
      CHAR = CHAR AND DELETE$MASK;      /* AND IT WITH 80 HEX */
      IF CHAR <> 0 THEN DELETE = TRUE;
END;

/* Print date in MM-DD-YY format. */
```

```
DATE:PROCEDURE;
     DCL MONTH LIT '25', DAY LIT '26', YEAR LIT '27';
      DECLARE DGT ADDR;
      LDSPC = FALSE;
      IF RFCB(MONTH) < 10 THEN CALL SPACE;
      DGTA = .DGT;
      DGT = RFCB(MONTH); CALL OUTDEC;
      CHAR = '-'; CALL PUTCHR;
      DGT = RFCB(DAY); CALL OUTDEC;
      CHAR = '-'; CALL PUTCHR;
      DGT = RFCB(YEAR); CALL OUTDEC;
      IF RFCB(DAY) < 10 THEN CALL SPACE;
      CALL SPACE;
END;

/* Open system record to allow reading of disk number
   and disk name. */

OPEN$SYSTEM:PROCEDURE;
      FCBA = .RFCB;
      RFCB(XFC) = QOS14R;
      RFCB(XUN) = WORKDRIVE;
      CALL FMS;
      IF ERROR THEN CALL HANDLE$ERRORS;
END;

/* The next routine returns with next record from the directory
   in RFCB.  On exit REOF = TRUE if end of file, else REOF = FALSE.

   FMS returns with ERROR = FALSE if ok, ERROR = TRUE if
   there has been a disk problem.  */

GET$INFO$RECORD:PROCEDURE;

      REOF = TRUE;
      FCBA = .RFCB;
      RFCB(XFC) = QRDIR;
      RFCB(XUN) = WORKDRIVE;
      CALL FMS;

      IF ERROR THEN DO;

                  IF RFCB(XES) = EEOF THEN RETURN;
                  ELSE CALL HANDLE$ERRORS;

                  END;

      REOF = FALSE;

END;

/* Open directory to allow reading into RFCB */

OPEN$DIRECTORY: PROCEDURE;
      FCBA = .RFCB;
      RFCB(XFC) = QOD4R;
      RFCB(XUN) = WORKDRIVE;
      CALL FMS;
      IF ERROR THEN CALL HANDLE$ERRORS;
END;

/* Print filename, extension, data created, disk name
   and disk number in columnar format. */

WRITE$INFORMATION: PROCEDURE;

      IF DELETE THEN RETURN;
      IF RFCB(XFN) = 0 THEN RETURN;

      CALL WRITE$FILENAME;
      CALL SPACE;
      CALL WRITE$EXTENSION;
      CALL SPACE;
      CALL DATE;
      CALL SPACE;
      CALL WRITE$DISK$NAME;
      CALL SPACE;
      CALL WRITE$DISK$NUMBER;
      CALL PCRLF;

END;

/* Write name of current disk into a buffer called DISKNAME
   for later printing. */

TRANSFER$DISK$NAME: PROCEDURE;
     BUF1 = .RFCB + 4;
     BUF2 = .DISKNAME;
     BUFSIZE = 8;
       DO WHILE BUFSIZE <> 0;
       CHAR = MEM(BUF1);
       IF CHAR <> 0 THEN
       MEM(BUF2) = MEM(BUF1);
       ELSE MEM(BUF2) = ' ';
       BUF1 = BUF1 + 1; BUF2 = BUF2 + 1;
       BUFSIZE = BUFSIZE -1;
       END;
    BUF3 = .RFCB + 19;
    BUF4 = .DISKNUMBER;
    MEMA(BUF4) = MEMA(BUF3);
END;

/* Output selected directory information. */

TRANSFER$DIRECTORY: PROCEDURE;

      CALL OPEN$DIRECTORY;
      CALL GET$INFO$RECORD;

      DO WHILE NOT REOF;
                  CALL DELETE$TEST;
                  CALL WRITE$INFORMATION;
                  CALL GET$INFO$R CORD;
      END;
END;
```

```
/* Read name of disk and number of disk from system
   information record */

READ$NUMBER: PROCEDURE;
      CALL OPEN$SYSTEM;
      CALL GET$INFO$RECORD;
      CALL TRANSFER$DISK$NAME;
END;

/*  MAIN */

DECLARE PROMPT DATA ('CHANGE DISK IN DRIVE  1 THEN HIT A KEY',4);
DECLARE PROMPT1 DATA ('TYPE AN "E" TO STOP. ', 04);
0AC22H: DECLARE OUTPUT$SWITCH BYTE;

STOPCUE = 0;
ENDFLAG = 'E';
DO WHILE  STOPCUE <> ENDFLAG;
    CALL READ$NUMBER;
    CALL TRANSFER$DIRECTORY;
     OUTPUT$SWITCH = TRUE;
      MSGA = .PROMPT;  CALL PSTRNG;
      MSGA = .PROMPT1; CALL PSTRNG;
    CALL GETCHR;
     OUTPUT$SWITCH = FALSE;
    STOPCUE = CHAR;
END;

CALL WARMS;

EOF
```

Dale Puckett
14753 Endsley
Woodbridge, VA 22193

BASIC UTILITY PACKAGE

This month we review six utility programs designed to make the life of the BASIC programmer a little easier. They are offered by Star-Kits, P. O. Box 2909, Mt. Kisco, N. Y. 10549. They were written by Peter Stark who has a very good touch with 6800 hardware and software. One of the programs is written in assembly language, the others are coded in BASIC.

The utilities are: BASEDIT, an editor designed mainly to renumber BASIC programs; PRETTY, a pretty-printer; VINDEX, a program which indexes variables; BACOMP, a utility which lists the differences between two BASIC programs; SHORTS, which shortens listings and speeds up execution of some programs; and BENTER, which automatically generates line numbers and puts a program on a disk. A bonus utility called FLOGEN also comes with the package. Given enough time it will print a flowchart of a BASIC program, pointing out all FOR-NEXT loops and transfers of control.

The packages are available in three versions: MF runs on the Mini-Flex DOS and SWTPC Disk Basic Version 3.0; F2 runs with Flex 2.0 and TSC Disk Basic; and PD runs with a Percom LFD-400 disk, Minidos-PLUSX DOS and Percom Super Basic. Some of the programs require 32K of memory although they can be modified to use less memory.

BASEDIT, the assembly language program is supplied in both source and object form. It was designed specifically to renumber BASIC programs and changes all GOTO's, GOSUB's, etc., within the program while it is changing the line numbers.

BASEDIT will prompt you for a starting line number. If you do not give one, it will start numbering lines at 1000. Unfortunately however, it will only increment line numbers by 10. This may seem like a shortcoming to the programmer who loves to remove all the bugs and then renumber his program with a line increment of one in order to make it hard to change.

BASEDIT is menu driven and also allows you to do minor editing to the BASIC file. It includes function to both (F)ind and (R)eplace strings. They both seem to work although editing with a full size editor is obviously more efficient. Both work on every occurance of the string in the program and the user should proceed cautiously. Stark says in his well written documentation that BASEDIT might take several minutes to renumber a long program. Yet, it seems to be fairly fast and I timed it at 14 seconds on a program 32 sectors long.

PRETTY, the pretty-printing program does several things. It separates the listing into pages, providing a page heading complete with date and page number on each sheet. It double-spaces before and after all REMs, breaking the listing into easily readable blocks. If a REM is encountered within a line it is automaticaly placed at column 50.

PRETTY indents each statement in a FOR-NEXT loop thereby illustrating the range of the loop. Nested loops are indented further and a very readable program results.

The program prompts you for the Port number, the number of lines on a page, the program name, and the date. It seems a shame that an option to control the width of the listing was not provided. There was no problem listing a program on my IBM typewriter, but when I tried one on my Model 15 teletype I occasionally ran out of space. I do not like to make routine listings on my IBM because it costs $30 just to get an estimate for repair.

When using the mini-FLEX version you must change line 180 so it will know which file to open for read. It you do not make this change before running you will get an error message which points to the line, but, it seems it would be much nicer asthetically if the author had prompted you to make the change and then type "CONT."

VINDEX prints a list which gives every line number where a variable is used. It is one of the most useful programs in the package. Its only fault appears to be its speed. I timed it at 11 minutes and 47 seconds before it started printing on the same 32 sector program mentioned earlier. It then took another four minutes to print the results on a CRT running at 60 characters per second. Most of this problem is caused by the fact that it is running in the original SWTPC BASIC. Here's the good news. On the documentation a stamp noted that an experimental version of VINDEX.CMD was enclosed on the disk. I'm pretty sure that VINDEX.CMD was compiled on the ABASIC compiler. It is fast and indexed the same 32 sector program in a little over 35 seconds, start to finish.

BACOMP is a utility that will help you find the changes you made in later versions of your BASIC programs. It reads from two separate BASIC program files and prints only the differences. Every time it encounters a line that is different it prints the line from one file and then indents and prints the same line from the other file.

SHORTS is a program you can run on your BASIC masterpiece after you have removed all the bugs. It shortens the program by removing all remarks that are not on lines referenced elsewhere and by concatentating several short lines into one. It is also possible to have it print you a list of all program transfers sorted by destination because the program needs that information before it can remove any REMs.

SHORTS is also hampered by the lack of speed of SWTPC BASIC. It took it just over 13 minutes to run on PRETTY, a BASIC source file which is 28 sectors long. But, the new file was only 16 sectors long.

BENTER is a short program and seems to perform as expected. It faithfully generated automatic line numbers while I typed in a short BASIC test program. And, it allowed me to pick the starting line number and the line increment. The resulting disk file loaded into BASIC and ran perfectly.

FLOGEN, the bonus program is also interesting. It reads a BASIC program from a disk file and prints it in an abbreviated form with arrows and lines connecting segments of code which go together. It also connects each NEXT with the proper FOR. It does not illustrate transfers caused by a GOSUB however since they almost always involve long transfers from one end of a program to the other. FLOGEN also runs slowly and it takes nearly 30 minutes to print a chart of VINDEX.BAS. VINDEX is 37 sectors long.

CONCLUSIONS

For the person who spends most of his time writing and debugging BASIC programs, this Basic Utility Package should be well worth the money. For those still using SWTPC type basics BASEDIT should be a great help. PRETTY is a program every BASIC programmer should be required to use. What good is a program if you can't read it six months later? PRETTY will go a long way toward improving your readability problems. VINDEX will shorten a lot of headaches, espcially for those programmers who write long BASIC programs with many variables. You probably couldn't find an easier way to keep track of them. SHORTS will help you out by improving the speed of execution. Just make sure you save a copy of the original program with all the REMs.

The numbering and formatting are almost automatic and the result is extremely pleasing to the eye. Readability is the answer to many programming problems and Stark is making it possible for you to let the computer do this housekeeping chore while you worry about the problem your solving. Isn't that why we use computers?

A 68 Micro Journal lab rating of AAA

Rating Scale:
AAA - Excellent
AA - Good
A - Fair (could be better but works)
P - Poor (may not always work properly)
X - Not recommended for children
(or anything else!)

ADDITIONAL COMMENTS

A few additional words regarding the review of our (AAA Chicago Computer Center) Basic K. The version that Jeff Craig reviewed is the disc version. We don't have separate versions for cassette and disc. Our Basics are popular because the cassette features permit users to transfer cassette files to disc (Smoke Signal Broadcasting) when the user adds disc to his system. Basic K has been superseded by Basic R.2 which in turn has been superseded by Basic R.3. All versions support cassette as well as Smoke Signal Disc. The version K supports sequential files whereas both R versions support sequential files (both space compressed and non space compressed) and random disc files (both byte orientated and record orientated).

All registered purchasers of Basic K and Basic R.2 are entitled to the manual for Basic R.3 free of charge. Just drop us a line with your name and address. All registered purchasers of Basic K and Basic R.2 can purchase Basic R.3 for $10.00. Specify whether you want it on disc or cassette. Source of Basic R.3 is not supplied, but initialization instructions are included with a listing of the parameter and jump tables so that the user has full ability to adapt Basic R.3 to his monitor, his choice of control characters, as well as to his choice of SSB DOS 3.1-4.2 ($6-7000, $A-B000, $C-D000) and DOS 5.1 ($6-7000, $A-B000, $C-D000).

32K of RAM is not required as indicated in the review. The earlier Basics use the SSB add "B" register to "X" register subroutine located in the DOS. The user is free to substitute his own equivalent routine elsewhere. Basic R.3 has its own self contained routine for this purpose.

MODEM PROGRAM FOR 68XX

AAA Chicago Computer Center announces the availability of our Modem Program Version 2.0. This program permits the user to interface a modem through a serial interface installed in any I/O port. Disc file transmission in both directions is supported as well as keyboard transmission. The routine that polls disc, keyboard, and modem is under complete user control. Disc system does not have to be DMA. Cost of instructions and source listing is $25.00. Add $10.00 for cassette or disc and be sure to specify monitor (GMXBUG, Smartbug, SWTBUG) and DOS (SSB, Mini Flex, Flex 2.0, Flex09).

AAA Chicago Computer Center
120 chestnut Lane
Wheeling, IL 60090

(312) 459-0450

* DISK BARGAIN *

Technical Systems Consultants, Inc, Box 2574, West Lafayette, Indiana, 47906, (317) 463-2502, Has available approximately 300-400 16 hole (sector), hardsector 5 1/4 inch minidisk at a very reasonable price. For those users using this disk format it is a fine opportunity to buy a supply of disk at a very good price (below wholesale).

Interested parties should call or write either Dave or Don at the address shown above.

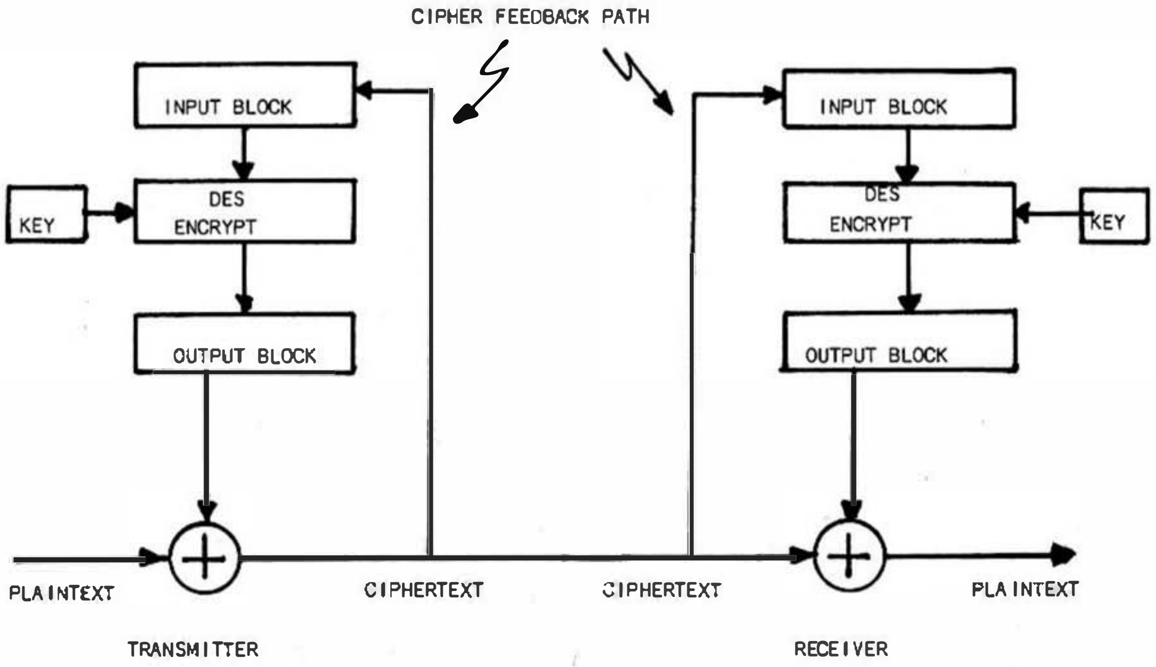
A 6800 SOFTWARE IMPLEMENTATION OF DATA STREAM ENCRYPTION BY THE DATA ENCRYPTION STANDARD

S. J. LaCour and T. F. Elbert
The University of West Florida
Pensacola, Florida 32504

The National Bureau of Standards has established its Data Encryption Standard (DES) as the single method by which encryption of non-classified data within all federal agencies is accomplished. The standard specifies a hardware implementation, and several manufacturers now market encryption modules which have been validated by the National Bureau of Standards. Those individuals with a requirement for data encryption not involving any federal agency, or those merely wishing to experiment with data encryption, can use a software version of the DES algorithm.

A previous article (Reference 1) has described a 6800 assembly language implementation of the DES algorithm which will run in 1100 bytes of memory, and which accomplishes 64-bit block encryption in accordance with the provisions of the basic algorithm. When applied to data stream encryption, i.e., the serial flow of data by bit or by character, block encryption methods are not desirable because the first bit or character cannot be encrypted until the last bit or character has been received. One alternative is to pad the block with an arbitrary pattern in order to fill all 64 bits, but this has the disadvantage that the plain text must then be stripped of the additional characters after decryption. The purpose of this article is to describe a 6800 software implementation of data stream encryption by means of a technique called cipher feedback.

The basic concept of cipher feedback is shown in the figure below. The reader will immediately note that there is no decryption used, but rather that synchronized encryption processes take place at both the transmitter and receiver. These encryption processes are used to generate a stream of "encrypting bits", which are then exclusive ORed with the plaintext bits at the transmitter to generate the ciphertext bits, and again with the ciphertext bits at the receiver to recover the plaintext. Since the exclusive OR is its own inverse transformation, the system works if the encrypting bits are synchronized between transmitter and receiver. The encrypting bits themselves are produced by a standard 64 bit DES block encryption, operating on the input block to produce the output block from which the encryption bits are obtained.



DATA STREAM ENCRYPTION BY THE CIPHER FEEDBACK MODE

Any number of bits from the output block may be used, and the remainder discarded. In one extreme, only a single bit would be used from each encrypted output block, and in the other all 64 bits would be used. The system described here is intended to encode the ASCII character set, as entered from a keyboard, making the use of eight bits from each output block the natural choice since the plaintext is grouped as eight bit blocks.

Synchronization of the input blocks at the transmitter and receiver is obtained by using the same key at both the transmitter and receiver locations, and the ciphertext itself as the input block. Hence the term "cipher feed back" as applied to this mode of operation. Since the ciphertext is common to both transmitter and receiver, synchronization is assured in the absence of any transmission errors. Even if a transmission error does occur, the system will automatically re-synchronize after 64 bits of ciphertext has been correctly received (eight ASCII characters in the implementation under consideration here). The same feature produces automatic synchronization on start-up after transmission of 64 bits, even if the initial input blocks are not identical.

In addition to self-synchronization, the cipher feedback mode of operation provides protection against a particular type of intrusion called "spoofing". This occurs when an intruder, while not knowing the secret key, does have knowledge of certain corresponding ciphertext and plaintext combinations. These could be obtained by observing the ciphertext resulting from a known plaintext. The intruders objective is to intercept certain portions of the ciphertext, alter it in accordance with the known ciphertext-plaintext combinations, and re-transmit it in such a manner that the intrusion goes undetected. Protection against spoofing is obtained by a feature called "garble extension", whereby if any portion of the ciphertext becomes garbled, the decryption of a certain amount subsequent ciphertext is also garbled. That the cipher feedback mode does provide garble extension is easily seen, since modification of any ciphertext between transmitter and receiver will immediately cause a loss of synchronization for the following 64 bits. In the system described here, garble or spoofing will produce a garbled decryption for the following eight ASCII characters.

The implementation described here was programmed for a SWTP 6800 microcomputer, and was made into a utility for the FLEX operating system. It utilizes an encrypt and a decrypt mode, permitting it to perform either the transmitter or receiver function. In either case, it responds to keystroke entries on the terminal, either encrypting a single keystroke entry into two hexadecimal digits or decrypting two hexadecimal digits into the appropriate ASCII character. Characters supported by the program are the set of ASCII keyboard characters.

The program has been configured as a FLEX utility, with the name DESSER1. It uses the DES algorithm presented in Reference 1, with the exception of the main routine. There are, of course, additional storage definitions required. The result is that this program is identical to that described in Reference 1 after line number 318, beginning with subroutine SHIFT.

The data stream encryption algorithm implemented here performs encryption in one byte increments, as described above. However, in order to increase the utility of the program for the user experimenting with data encryption, the input data, whether it be plaintext or ciphertext, is stored in an 80 byte buffer during the input operation. The encryption or decryption process is then initiated by a carriage return command, at which time the buffer contents are treated as a data stream. The first eight bytes in the data stream are used for synchronization of the receiver, so that actual decryption begins with the ninth character sent.

The use of this buffer really defeats the purpose of the data stream encryption concept; it is used here merely as an aid in simulating the data stream. In an actual application of data stream encryption, the reading of the buffer would be replaced by the data stream itself.

Upon the DESSER1 request to FLEX, the program responds with a request for specification of transmitter or receiver (encrypt or decrypt) mode. When this is provided, it will request the key, which is a string of 16 hexadecimal digits. When the key is typed in, the program will request the input data, which cannot exceed 80 characters for plaintext or 80 bytes for ciphertext, to be followed by a carriage return. The encrypted or decrypted message is then written out in a data stream mode as it is created by the program. The program is not fast, since the complete DES algorithm must be exercised for each character encrypted or decrypted. For a one megahertz system, a typical speed of two cherecters per second has been observed.

Following the listing is shown a set of examples using the FLEX utility program DESSERT. The first and second show an encryption followed by e decryption. The first eight characters, SYNCWORD, are not properly deciphered, since they constitute the synchronizing bytes. Example 3 shows an attempt at decyphering using the wrong key, while example 4 shows the garble extension and re-synchronization features (a cerriage return was generated in the resulting garble in this particular case). The remaining examples illustrate the capability for lower case and other ASCII symbols.

For use in experimenting with data stream encryption, there are several warm start addresses which permit entry into the program without recourse to FLEX. These are 02FE, which initiates the entire sequence just as if a FLEX access had been made; 0338, which encrypts plaintext with the currently existing key; and 0399, which decrypts ciphertext with the currently existing key.

REFERENCES

1. LaCour, S. J. and Elbert, T. F., "A Software Data Encryption Standard for the 6800."

2. Campbell, Carl M., "Design and Specificetion of Cryptographic Capabilities," IEEE Communications Society Magazine, November, 1978.

```
11/07/79            1.DESDIR1.OUT                  PAGE 01

                                  NAM    DES
                      ****************************************************
                      *     STANDARD DATA ENCRYPTION ALGORITHM           *
                      ****************************************************
                      *        AUTHOR: Sam J. La Cour Jr.                *
                      ****************************************************
       0000                       ORG    $0000
       0000 20 01                 BRA    ENT
       0002 01          VN        FCB    1
       0003 7E 02 FE    ENT       JMP    MAIN
       0006 49          KEYMSG    FCC    /INPUT KEY IN BYTES:/
       0019 04                    FCB    $04
       001A             STLOC     RMB    2
       001C             NRESBT    RMB    1
       001D             NPERM     RMB    1
       001E             AINPUT    RMB    2
       0020             ARESUD    RMB    2
       0022             TABADR    RMB    2
       0024             NBIT      RMB    1
       0025             RD        RMB    1
       0026             TMASK     RMB    1
       0027             L         RMB    4
       002B             R         RMB    4
       002F             K         RMB    8
       0037             DS        RMB    4
       003B             RESWD     RMB    8
       0043             INPUT     RMB    8
         4B             C         RMB    4
```

```
       004F             D         RMB    4
       0053             SDIR      RMB    1
       0054 3F          SNUM      FCB    $3F,$7E
       0056 00 2F       KADR      FDB    K
       0058 00 37       K1ADR     FDB    K1
       005A 01 BE       SADR      FDB    S
       005C             KEY       RMB    8
       0064             APARM     RMB    2
       0066             JFLAG     RMB    1
       0067             PLAIN     RMB    1
       0068             NADR1     RMB    2
       006A             NADR2     RMB    2
       006C             CIPHER    RMB    1
       006D             INPUT1    RMB    8
       0075             INPUT2    RMB    8
       007D             SAV       RMB    1
       007E             KEY1      RMB    8
       0086             KEY2      RMB    8
       AD21             CLASS     EQU    $AD21
       008E 00          PARM0     FCB    $00,$00
       0090 00 43                 FDB    INPUT,L,INITAB
       0096 00                    FCB    $00,$04
       0098 00 5C                 FDB    KEY,C,PCH1A
       009E 00                    FCB    $30,$04
       00A0 00 5C                 FDB    KEY,D,PCH1B
       00A6 06          KPARM     FCB    $06,$08
       00A8 00 4B                 FDB    C,K,PCH2
       00AE 06                    FCB    $06,$08
       00B0 00 2B                 FDB    R,RESWD,E

                        1.DESDER1.OUT

       00B6 00                    FCB    $08,$04
       00B8 00 37                 FDB    K1,RESWD,P
       00BE 08                    FCB    $08,$08
       00C0 00 27                 FDB    L,RESWD,INVTAB
       00C6 27          INITAB    FCB    $27,$26,$25,$24,$23,$22,$21,$20
       00CE 47                    FCB    $47,$46,$45,$44,$43,$42,$41,$40
       00D6 67                    FCB    $67,$66,$65,$64,$63,$62,$61,$60
       00DE 07                    FCB    $07,$06,$05,$04,$03,$02,$01,$00
       00E6 17                    FCB    $17,$16,$15,$14,$13,$12,$11,$10
       00EE 37                    FCB    $37,$36,$35,$34,$33,$32,$31,$30
       00F6 57                    FCB    $57,$56,$55,$54,$53,$52,$51,$50
       00FE 77                    FCB    $77,$76,$75,$74,$73,$72,$71,$70
       0106 17          PCH1A     FCB    $17,$16,$15,$14,$13,$12,$11
       010D 10                    FCB    $10,$27,$26,$25,$24,$23,$22
       0114 21                    FCB    $21,$20,$37,$36,$35,$34,$33
       011B 32                    FCB    $32,$31,$30,$47,$46,$45,$44
       0122 77          PCH1B     FCB    $77,$76,$75,$74,$73,$72,$71
       0129 70                    FCB    $70,$67,$66,$65,$64,$63,$62
       0130 61                    FCB    $61,$60,$57,$56,$55,$54,$53
       0137 52                    FCB    $52,$51,$50,$43,$42,$41,$40
       013E 61          PCH2      FCB    $61,$12,$31,$82,$10,$50
       0144 30                    FCB    $30,$43,$71,$60,$52,$21
       014A 72                    FCB    $72,$32,$41,$40,$23,$80
       0150 81                    FCB    $81,$70,$33,$42,$51,$20
       0156 55                    FCB    $55,$86,$34,$15,$36,$37
       015C 24                    FCB    $24,$45,$76,$16,$54,$46
       0162 85                    FCB    $85,$56,$35,$47,$64,$17
       0168 26                    FCB    $26,$65,$66,$84,$14,$44
       016E 83          E         FCB    $83,$10,$20,$30,$40,$50
       0174 40                    FCB    $40,$50,$60,$70,$80,$11
       017A 80                    FCB    $80,$11,$21,$31,$41,$51
       0180 41                    FCB    $41,$51,$61,$71,$81,$12
       0186 81                    FCB    $81,$12,$22,$32,$42,$52
       018C 42                    FCB    $42,$52,$62,$72,$82,$13
       0192 82                    FCB    $82,$13,$23,$33,$43,$53
       0198 43                    FCB    $43,$53,$63,$73,$83,$10
       019E 81          P         FCB    $81,$70,$42,$52
       01A2 53                    FCB    $53,$41,$43,$12
       01A6 10                    FCB    $10,$71,$72,$23
       01AA 50                    FCB    $50,$22,$73,$21
       01AE 20                    FCB    $20,$80,$82,$61
       01B2 83                    FCB    $83,$33,$30,$11
       01B6 32                    FCB    $32,$31,$63,$60
       01BA 62                    FCB    $62,$31,$40,$13
       01BE E4          S         FCB    $E4,$D1,$2F,$B8,$3A,$6C,$59,$07
       01C6 0F                    FCB    $0F,$74,$E2,$D1,$A6,$CB,$95,$38
       01CE 41                    FCB    $41,$E8,$D6,$2B,$FC,$97,$3A,$50
       01D6 FC                    FCB    $FC,$82,$49,$17,$5B,$3E,$A0,$6D
       01DE F1                    FCB    $F1,$8E,$6B,$34,$97,$2D,$C0,$5A
       01E6 3D                    FCB    $3D,$47,$F2,$8E,$C0,$1A,$69,$B5
       01EE 0E                    FCB    $0E,$7B,$A4,$D1,$58,$C6,$93,$2F
       01F6 D8                    FCB    $D8,$A1,$3F,$42,$B6,$7C,$05,$E9
       01FE A0                    FCB    $A0,$9E,$63,$F5,$1D,$C7,$B4,$28
       0206 D7                    FCB    $D7,$09,$34,$6A,$28,$5E,$CB,$F1
       020E D6                    FCB    $D6,$49,$8F,$30,$B1,$2C,$5A,$E7
       0216 1A                    FCB    $1A,$D0,$69,$87,$4F,$E3,$B5,$2C
       021E 7D                    FCB    $7D,$E3,$06,$9A,$12,$85,$BC,$4F
       0226 D8                    FCB    $D8,$B5,$6F, 03,$47,$2C,$1A,$E9

                        1.DESDCR1.OUT

       022E A6                    FCB    $A6,$90,$CB,$7D,$F1,$3E,$52,$84
       0236 3F                    FCB    $3F,$06,$A1,$D8,$94,$5B,$C7,$2E
       023E 2C                    FCB    $2C,$41,$7A,$B6,$85,$3F,$D0,$E9
       0246 EB                    FCB    $EB,$2C,$47,$D1,$50,$FA,$39,$86
       024E 42                    FCB    $42,$1B,$AD,$78,$F9,$C5,$63,$0E
       0256 B8                    FCB    $B8,$C7,$1E,$2D,$6F,$09,$A4,$53
       025E C1                    FCB    $C1,$AF,$92,$68,$0D,$34,$E7,$5B
       0266 AF                    FCB    $AF,$42,$7C,$95,$61,$DE,$0B,$38
       026E 9E                    FCB    $9E,$F5,$28,$C3,$70,$4A,$1D,$B6
       0276 43                    FCB    $43,$2C,$95,$FA,$BE,$17,$60,$8D
       027E 4B                    FCB    $4B,$2E,$F0,$8D,$3C,$97,$5A,$61
       0286 D0                    FCB    $D0,$B7,$49,$1A,$E3,$5C,$2F,$86
       028E 14                    FCB    $14,$BD,$C3,$7E,$AF,$68,$05,$92
       0296 6B                    FCB    $6B,$D8,$14,$A7,$95,$0F,$E2,$3C
       029E D2                    FCB    $D2,$84,$6F,$B1,$A9,$3E,$50,$C7
       02A6 1F                    FCB    $1F,$D8,$A3,$74,$C5,$6B,$0E,$92
       02AE 7B                    FCB    $7B,$41,$9C,$E2,$06,$AD,$F3,$58
       02B6 21                    FCB    $21,$E7,$4A,$8D,$FC,$90,$35,$6B
       02BE 84          INVTAB    FCB    $84,$80,$85,$81,$86,$82,$87,$83
       02C6 74                    FCB    $74,$70,$75,$71,$76,$72,$77,$73
       02CE 64                    FCB    $64,$60,$65,$61,$66,$62,$67,$63
       02D6 54                    FCB    $54,$50,$55,$51,$56,$52,$57,$53
       02DE 44                    FCB    $44,$40,$45,$41,$46,$42,$47,$43
       02E6 34                    FCB    $34,$30,$35,$31,$36,$32,$37,$33
       02EE 24                    FCB    $24,$20,$25,$21,$26,$22,$27,$23
       02F6 14                    FCB    $14,$10,$15,$11,$16,$12,$17,$13
```

```
E1AC            INEEE   EQU   $E1AC
E1D1            OUTEEE  EQU   $E1D1
E0BF            OUT2H   EQU   $E0BF
E0C8            OUT4HS  EQU   $E0C8
E047            BADDR   EQU   $E047
E055            BYTE    EQU   $E055
E07E            PDATA1  EQU   $E07E
E0CA            OUT2HS  EQU   $E0CA
E0CC            OUTS    EQU   $E0CC
AD03            WARMS   EQU   $AD03
AD1E            PSTRNG  EQU   $AD1E
AD2D            GETFIL  EQU   $AD2D
AD33            SETEXT  EQU   $AD33
AD36            ADDBX   EQU   $AD36
AD3F            RPTERR  EQU   $AD3F
AD15            GETCHR  EQU   $AD15
AD24            PCRLF   EQU   $AD24
AD18            PUTCHR  EQU   $AD18
02FE BD AD 24   MAIN    JSR   PCRLF
0301 86 00              LDA A #$00
0303 CE 04 70           LDX   #TRMSG
0306 BD E0 7E           JSR   PDATA1
0309 BD AD 24           JSR   PCRLF
030C BD E0 55           JSR   BYTE
030F B7 04 D4           STA A MODE        MODE = 00 for transmitter, 01 for reciever.
0312 BD AD 24           JSR   PCRLF
0315 CE 00 06           LDX   #KEYMSG
0318 BD E0 7E           JSR   PDATA1
031B BD AD 24           JSR   PCRLF
031E C6 08              LDA B 08
0320 CE 00 5C           LDX   #KEY
0323 37         INKEY   PSH B

                   1.DESSER1.OUT

0324 BD E0 55           JSR   BYTE
0327 A7 00              STA A 0,X
0329 08                 INX
032A 33                 PUL B
032B 5A                 DEC B
032C 2E F5              BGT   INKEY
032E BD AD 24           JSR   PCRLF
0331 B6 04 D4           LDA A MODE
0334 81 00              CMP A #$00        If reciever then go to decode cycle
0336 26 61              BNE   PCPHSG
0338 CE 00 6D           LDX   #INPUT1     Move transmitters input block to DES input block
033B DF 68              STX   MADR1
033D CE 00 43           LDX   #INPUT
0340 DF 6A              STX   MADR2
0342 86 08              LDA A 08
0344 BD 04 E8           JSR   MOVE
0347 CE 04 93           LDX   #PTXTMS
034A BD E0 7E           JSR   PDATA1
034D BD AD 24           JSR   PCRLF
0350 7F 04 6F           CLR   NBUF
0353 CE 04 1D           LDX   #BUFFER
0356 BD AD 15   GETPLN  JSR   GETCHR      Input a character from keyboard
0359 81 0D              CMP A #$0D        A <CR> signals end of input
035B 27 08              BEQ   GOTPLN
035D A7 00              STA A 0,X         Store the character in the buffer
035F 08                 INX
0360 7C 04 6F           INC   NBUF
0363 20 F1              BRA   GETPLN
0365 7C 04 6F   GOTPLN  INC   NBUF
0368 A7 00              STA A 0,X         Also store the <CR> in the buffer
036A BD AD 24           JSR   PCRLF
036D CE 04 1D           LDX   #BUFFER
0370 FF 04 6D           STX   BUFADR
0373 BD 05 3F   ENCODE  JSR   ITER        Use DES to get an output block (RESWD)
0376 FE 04 6D           LDX   BUFADR
0379 A6 00              LDA A 0,X         Plaintext byte from buffer
037B 08                 INX
037C FF 04 6D           STX   BUFADR
037F 98 3D              EOR A RESWD       Form ciphertext byte
0381 97 7D              STA A SAV
0383 CE 00 7D           LDX   #SAV
0386 BD E0 CA           JSR   OUT2HS      Output ciphertext byte to terminal
0389 96 7D              LDA A SAV
038B CE 00 43           LDX   #INPUT
038E BD 04 D5           JSR   ROTATE      Feed ciphertext byte into DES input block
0391 7A 04 6F           DEC   NBUF
0394 2E DD              BGT   ENCODE      Keep going thru buffer
0396 7E AD 03           JMP   WARMS       Return to FLEX
0399 CE 04 B3   PCPHSG  LDX   #CPTXTM     Begin decipher coding
039C BD E0 7E           JSR   PDATA1
039F BD AD 24           JSR   PCRLF
03A2 CE 00 75           LDX   #INPUT2
03A5 DF 68              STX   MADR1
03A7 CE 00 43           LDX   #INPUT
03AA DF 6A              STX   MADR2
03AC 86 08              LDA A 8
03AE BD 04 E8           JSR   MOVE        Move reciever's input block to DES input block
03B1 7F 04 6F           CLR   NBUF

                   1.DEDOED1.OUT

03B4 CE 04 1D           LDX   #BUFFER
03B7 FF 04 6D           STX   BUFADR
03BA BD AD 15   GETCPH  JSR   GETCHR      Input format for decode is 2 hex characters
03BD 81 0D              CMP A #$0D        so that a <CD> can be used to mark the end
03BF 27 34              BEQ   GOTCPH
03C1 81 39              CMP A #$39        But need to convert 2 chars into 1 byte
03C3 2F 06              BLE   NUMER
03C5 84 0F              AND A #$0F
03C7 8B 09              ADD A #9
03C9 20 02              BRA   ADJ
03CB 84 0F      NUMER   AND A #$0F
03CD 48         ADJ     ASL A
03CE 48                 ASL A
03CF 48                 ASL A
03D0 48                 ASL A
03D1 97 7D              STA A SAV
03D3 BD AD 15           JSR   GETCHR
03D6 81 39              CMP A #$39
03D8 2F 06              BLE   NUMER1
03DA 84 0F              AND A #$0F
03DC 8B 09              ADD A #9
03DE 20 02              BRA   NOADJ
03E0 84 0F      NUMER1  AND A #$0F
03E2 9A 7D      NOADJ   ORA A SAV
03E4 FE 04 6D           LDX   BUFADR
03E7 A7 00              STA A 0,X         Put final converted byte in buffer
03E9 08                 INX
03EA FF 04 6D           STX   BUFADR
03ED 7C 04 6F           INC   NBUF
03F0 BD E0 CC           JSR   OUTS        Output a space for readability
03F3 20 C5              BRA   GETCPH
03F5 BD AD 24   GOTCPH  JSR   PCRLF
03F8 CE 04 1D           LDX   #BUFFER
03FB FF 04 6D           STX   BUFADR
03FE BD 05 3F   DECODE  JSR   ITER        Perform DES on input block
0401 FE 04 6D           LDX   BUFADR
0404 A6 00              LDA A 0,X         Get a byte from buffer
0406 08                 INX
0407 FF 04 6D           STX   BUFADR
040A CE 00 43           LDX   #INPUT      Feed the ciphertext that was input
040D BD 04 D5           JSR   ROTATE      back into input block
0410 98 3D              EOR A RESWD       Form plaintext
0412 BD E1 D1           JSR   OUTEEE      Output the plaintext
0415 7A 04 6F           DEC   NBUF
0418 2E E4              BGT   DECODE      Do it again
041A 7E AD 03           JMP   WARMS
041D            BUFFER  RMB   80
046D            BUFADR  RMB   2
046F            NBUF    RMB   1
0470 54         TRMSG   FCC   /TRANSMITTER OR RECIEVER (00 OR 01)/
0492 04                 FCB   $04
0493 49         PTXTMS  FCC   /INPUT PLAIN TEXT FOLLOWED BY CR/
04B2 04                 FCB   $04
04B3 49         CPTXTM  FCC   /INPUT CIPHER TEXT FOLLOWED BY CR/
04D3 04                 FCB   $04
04D4            MODE    RMB   1
04D5 36         ROTATE  PSH A Subroutine to rotate A into right side of
04D6 37                 PSH B             data block pointed to by X

                   1.DECODER1.OUT

04D7 36                 PSH A
04D8 C6 07              LDA B #7
04DA A6 01      ROT1    LDA A 1,X
04DC A7 00              STA A 0,X
04DE 08                 INX
04DF 5A                 DEC B
04E0 2E F8              BGT   ROT1
04E2 32                 PUL A
04E3 A7 00              STA A 0,X
04E5 33                 PUL B
04E6 32                 PUL A
04E7 39                 RTS
04E8 B7 05 01   MOVE    STA A NSTR        Subroutine to move one block of data to another
04EB DE 6A              LDX   MADR2
04ED DF 6A      LOOP1   STX   MADR2
04EF DE 68              LDX   MADR1
04F1 A6 00              LDA A 0,X
04F3 08                 INX
04F4 DF 68              STX   MADR1
04F6 DE 6A              LDX   MADR2
04F8 A7 00              STA A 0,X
04FA 08                 INX
04FB 7A 05 01           DEC   NSTR
04FE 26 ED              BNE   LOOP1
0500 39                 RTS
0501            NSTR    RMB   1
                ******************************************
                * SUBROUTINE SHIFT:                      *
                *                                        *
                *  FUNCTION: Rotates 28 bits pointed to by X   *
                *            either left or right depending on *
                *            the value in "MODE" (00 for left, *
                *            01 for right). The result is stored *
                *            in the source field.        *
                *                                        *
                *  EXTERNAL ROUTINES: NONE               *
                *                                        *
                ******************************************
0502 36         SHIFT   PSH A             Save accumulators A and B.
0503 37                 PSH B
0504 36                 PSH A             Clear out right side of last byte
0505 A6 03              LDA A 3,X         of C or D.
0507 84 F0              AND A #$F0
0509 A7 03              STA A 3,X
050B 32                 PUL A
050C 81 01              CMP A #$01        Encrypt or decrypt?
050E 27 15              BEQ   RSHIFT      Decrypt -- right rotate.
0510 68 03      LSHIFT  ASL   3,X         Encrypt -- left rotate.
0512 A6 03              LDA A 3,X
0514 69 02              ROL   2,X
0516 69 01              ROL   1,X
0518 69 00              ROL   0,X
051A 89 0F              ADC A #$0F        Force carry into bit 0 of last
051C 84 F0              AND A #$F0        byte and reset bits 3 - 0.
051E A7 03              STA A 3,X
0520 5A                 DEC B             More than one shift?
0521 2E ED              BGT   LSHIFT      Yes -- do it again.
0523 20 17              BRA   OUTSHF      Exit.

                   1.DESDER1.OUT

0525 64 00      RSHIFT  LSR   0,X         Decrypt -- right rotate.
0527 66 01              ROR   1,X
0529 66 02              ROR   2,X
052B 66 03              ROR   3,X
052D A6 03              LDA A 3,X
052F 48                 ASL A             Force previous bit 4 of last byte
0530 48                 ASL A             to position 1 so it can be placed
0531 48                 ASL A             in bit 1 of the first byte which
0532 48                 ASL A             has been cleared by the LSR 0,X.
0533 84 80              AND A #$80        Get this bit.
0535 AA 00              ORA A 0,X         OR it into byte 1.
0537 A7 00              STA A 0,X         Resave byte 1.
0539 5A                 DEC B             More than one shift?
053A 2E E9              BGT   RSHIFT      Yes -- do it again.
053C 33         OUTSHF  PUL B             Restore accumulators A and B.
053D 32                 PUL A
053E 39                 RTS
```

```
                     *************************************************
                     * SUBROUTINE ITER:                              *
                     *                                               *
                     *  FUNCTION: Actually the main DES routine. ITER *
                     *            calls all other DES routines and   *
                     *            directs logic flow for either      *
                     *            encryption or decryption.          *
                     *                                               *
                     * EXTERNAL ROUTINES: NONE                       *
                     *                                               *
                     *************************************************
053F CE 00 0E  ITER    LDX    #PARMS    Initialize parameter pointer to
0542 DF 1A             STX    STLOC     first group of parameters.
0544 CE 3F 7E          LDX    #$3F7E    Initialize shift schedule indicator.
0547 DF 54             STX    SNUM
0549 BD 05 DB          JSR    PERM      Perform initial permutation.
054C BD 05 DB          JSR    PERM      Perform permuted choice 1a.
054F BD 05 DB          JSR    PERM      Perform permuted choice 1b.
0552 86 10             LDA A  #16       Initialize iteration counter.
0554 36        ITER1   PSH A            Save current iteration counter.
0555 CE 00 A6          LDX    #EXPDBO   Force parameter pointer to start
0558 DF 1A             STX    STLOC     of parameters used in iterations.
055A 96 53             LDA A  EDIR      Get encrypt, decrypt mode.
055C 26 1B             BNE    DECR      If 1 then decrypt mode.
055E 78 00 55          ASL    SNUM+1    Rotate next shift schedule bit into
0561 79 00 54          ROL    SNUM      carry then add the carry to 1,
0564 C6 01             LDA B  #$01      giving the number of shifts.
0566 C9 00             ADC B  #$00
0568 CE 00 4B          LDX    #C        Prepare to shift C to the left.
056B BD 05 02          JSR    SHIFT     Shift C left.
056E CE 00 4F          LDX    #D        Prepare to shift D to the left.
0571 BD 05 02          JSR    SHIFT     Shift D left.
0574 BD 05 DB          JSR    PERM      Perform permuted choice 2.
0577 20 1A             BRA    GOON      Continue.
0579 8D 60     DECR    BSR    PERM      Perform permuted choice 2 first.
057B 96 53             LDA A  EDIR      Get encrypt, decrypt code.
057D 74 00 54          LSR    SNUM      Rotate next shift schedule bit into
0580 76 00 55          ROR    SNUM+1    carry ( from right for decryption ).
0583 C6 01             LDA B  #$01      Now add the carry to 1 giving the
0585 C9 00             ADC B  #$00      number of shifts.
0587 CE 00 4B          LDX    #C        Prepare to shift C to the right.

          1.DESSER1.OUT

058A BD 05 02          JSR    SHIFT     Shift C to the right.
058D CE 00 4F          LDX    #D        Prepare to shift D to the right.
0590 BD 05 02          JSR    SHIFT     Shift D to the right.
0593 8D 46     GOON    BSR    PERM      Perform E permutation on R.
0595 CE 00 3F          LDX    #K        Prepare to EOR subkey K with
0598 86 08             LDA A  #08       result of E permutation, result in K.
059A 36        GOT1    PSH A
059B A6 00             LDA A  0,X       Get a byte of K.
059D A8 0C             EOR A  12,X      EOR it with a byte of "INPUT".
059F A7 00             STA A  0,X       Store it back in K.
05A1 08                INX
05A2 32                PUL A
05A3 4A                DEC A
05A4 2E F4             BGT    GOT1
05A6 BD 06 3D          JSR    PERMS     Perform S1 - S8 selection mapping.
05A9 8D 30             BSR    PERM      Perform P permutation giving f(R,K).
05AB CE 00 27          LDX    #L        Prepare to swap and EOR L and R.
05AE 86 04             LDA A  #04       Four bytes in each.
05B0 36        GOT2    PSH A
05B1 A6 00             LDA A  0,X       Get a byte of L.
05B3 A8 14             EOR A  20,X      EOR L with f(R,K) ( in "INPUT" ).
05B5 E6 04             LDA B  4,X       Get a byte of R.
05B7 A7 04             STA A  4,X       Put new L where R was.
05B9 E7 00             STA B  0,X       Put R where L was.
05BB 08                INX
05BC 32                PUL A
05BD 4A                DEC A
05BE 2E F0             BGT    GOT2
05C0 32                PUL A            Pull iteration counter off stack.
05C1 4A                DEC A            Decrement it
05C2 2E 90             BGT    ITER1     Keep going.
05C4 CE 00 27          LDX    #L        Prepare to perform final swap.
05C7 86 04             LDA A  #4
05C9 36        SW1     PSH A
05CA A6 00             LDA A  0,X       Get a byte of L.
05CC E6 04             LDA B  4,X       Get a byte of R.
05CE E7 00             STA B  0,X       Put R in L.
05D0 A7 04             STA A  4,X       Put L in R.
05D2 08                INX
05D3 32                PUL A
05D4 4A                DEC A
05D5 2E F2             BGT    SW1
05D7 BD 05 DB          JSR    PERM      Perform inverse initial permutation.
05DA 39                RTS              Finished.
                     *************************************************
                     * SUBROUTINE PERM:                              *
                     *                                               *
                     *  FUNCTION: Performs bit mapping from input    *
                     *            to output using a mapping table.   *
                     *            Input,output and table are specified*
                     *            in a parameter table which is      *
                     *            traversed serially each time       *
                     *            PERM is called. The mapping table  *
                     *            entries are in the form mmmmbbbb   *
                     *            where mmmm is a mask number which  *
                     *            gives the location of the source   *
                     *            bit within the source byte.        *
                     *            The bits are numbered from 1 TO 8. *

          1.DESSER1.OUT

                     *            bbbb gives the location within the *
                     *            input of the byte which contains   *
                     *            the desired bit. These bytes are   *
                     *            numbered from 0 TO 7.              *
                     *            The parameter list also gives the  *
                     *            number of bytes and the byte length *
                     *            of the output.                     *
                     *                                               *
                     * EXTERNAL ROUTINES:                            *
                     *    FLEX: ADDBX                                *
                     *                                               *
                     *************************************************
05DB 86 06     PERM    LDA A  #6        Number of parameter bytes to be saved
05DD CE 00 1C          LDX    #CURENT   Address of parameter storage area.
05E0 DF 64             STX    APARM
05E2 DE 1A     P1      LDX    STLOC     Current parameter pointer.
05E4 E6 00             LDA B  0,X       Get a parameter byte.
05E6 08                INX              Move parameter pointer along.
05E7 DF 1A             STX    STLOC     Save it.
05E9 DE 64             LDX    APARM     Address of receiving field.
05EB E7 00             STA B  0,X       Move parameter to current parameter list.
05ED 08                INX              Move this pointer along also.
05EE DF 64             STX    APARM
05F0 4A                DEC A
05F1 26 EF             BNE    P1        Do until all parameters moved.
05F3 DE 22             LDX    TABADR    Get table pointer from parameter list.
05F5 D6 1C     PLOOP   LDA B  NRESBT    Number of bits per permuted byte.
05F7 D7 24             STA B  NBIT      Save it.
05F9 7F 00 25          CLR    RB        Clear work byte.
05FC E6 00     SHFIN   LDA B  0,X       Get byte from table.
05FE 17                TBA              Put a copy in A.
05FF 84 F0             AND A  #$F0      Extract mask for source bit.
0601 44                LSR A            Right justify mask number.
0602 44                LSR A
0603 44                LSR A
0604 44                LSR A
0605 DE 1E             LDX    AINPUT    Get input address from parameters.
0607 C4 0F             AND B  #$0F      Extract source byte number.
0609 BD AD 36          JSR    ADDBX
060C E6 00             LDA B  0,X       Get source byte.
060E 7F 00 26          CLR    TMASK     Prepare mask area.
0611 0D                SEC              Set carry to be rotated into mask.
0612 76 00 26  SLOOP   ROR    TMASK     Location of mask bit is in A.
0615 4A                DEC A
0616 2E FA             BGT    SLOOP     Continue shifting mask bit in.
0618 D4 26             AND B  TMASK     Use mask to get bit from source byte.
061A C9 FF             ADC B  #$FF      Any one bit will be forced into the carr
061C 79 00 25          ROL    RB        Move the carry into RB.
061F DE 22             LDX    TABADR    Move pointer to next table entry.
0621 08                INX
0622 DF 22             STX    TABADR
0624 7A 00 24          DEC    NBIT      Decrement bit counter.
0627 2E D3             BGT    SHFIN     Continue until finished with current by
0629 D6 25             LDA B  RB        Get completed work byte.
062B DE 20             LDX    ARESUO    Get result address.
062D E7 00             STA B  0,X       Store work byte.
062F 08                INX              Point to next result byte.
0630 DF 20             STX    ARESUO

          1.DESSER1.OUT

0632 7F 00 25          CLR    RB        Clear out work byte for next iteration.
0635 DE 22             LDX    TABADR    Get current table location.
0637 7A 00 1D          DEC    NPERM     Decrement byte counter.
063A 2E B9             BGT    PLOOP     Continue till finished.
063C 39                RTS              Return to sender.
                     *************************************************
                     * SUBROUTINE PERMS:                             *
                     *                                               *
                     *  FUNCTION: Performs S mapping from K to K1    *
                     *            using S tables which are compacted *
                     *            so that odd-even elements are in   *
                     *            the same byte.                     *
                     *                                               *
                     * EXTERNAL ROUTINES:                            *
                     *    FLEX: ADDBX                                *
                     *                                               *
                     *************************************************
063D C6 00     PERMS   LDA B  #00       Initialize select S iteration counter.
063F 37        PERMS1  PSH B
0640 DE 56             LDX    KADR      Get current address of K.
0642 E6 00             LDA B  0,X       Get a byte of K.
0644 17                TBA              Put a copy in A.
0645 C4 1E             AND B  #$1E      Mask bits 4 - 7 to get column no.
0647 54                LSR B            Right justify column number.
0648 D7 25             STA B  RB        Save column number for now.
064A 54                LSR B            Integer divide col. no. by 2.
064B 84 21             AND A  #$21      Get row number.
064D 8B 0F             ADD A  #$0F      Force bit 1 next to bit 3.
064F 84 F0             AND A  #$F0      Clear right part of byte.
0651 44                LSR A            Integer divide row number by 2.
0652 DE 5A             LDX    SADR      Get beginning address of current S table.
0654 1B                ABA              Form byte number within table.
0655 16                TAB
0656 BD AD 36          JSR    ADDBX
0659 E6 00             LDA B  0,X       Get table entry.
065B 76 00 25          ROR    RB        Rotate original col. no. to see if the
065E 24 07             BCC    SETL      value is on the left or right side.
0660 7F 00 26          CLR    TMASK     Clear flag byte.
0663 C4 0F             AND B  #$0F      Get the right value from the table entry.
0665 20 06             BRA    PSIDE     So see which side to put it in.
0667 C4 F0     SETL    AND B  #$F0      Get the left value from the table entry.
0669 86 01             LDA A  #01       Set left side flag.
066B 97 26             STA A  TMASK     Save it.
066D 32        PSIDE   PUL A            Get copy of iteration counter.
066E 36                PSH A            Restore it to stack.
066F 46                ROR A            Rotate rightmost bit into carry.
0670 24 13             BCC    STL       If 0 then even numbered iteration.
0672 96 26             LDA A  TMASK     Odd number - if the entry is already
0674 27 04             BEQ    RSWFD     on the right side then no shift needed.
0676 54                LSR B
0677 54                LSR B
0678 54                LSR B
0679 54                LSR B
067A DE 58     RSWFD   LDX    K1ADR     Get current byte address.
067C EA 00             ORA B  0,X       Place right side of byte in.
067E E7 00             STA B  0,X
0680 08                INX              Since the byte is now completed,
0681 DF 58             STX    K1ADR     increment the result byte pointer.

          1.DESSER1.OUT

0683 20 0C             BRA    STO
0685 96 26     STL     LDA A  TMASK     Get flag.
0687 26 04             BNE    NOMFL     If flag is 0 then need to left
0689 58                ASL B            justify the table entry to fit
068A 58                ASL B            the output.
068B 58                ASL B
068C 58                ASL B
068D DE 58     NOMFL   LDX    K1ADR     Get current result byte address.
068F E7 00             STA B  0,X       Left side -- just store it.
0691 DE 56     STO     LDX    KADR      Increment source data addr pointer.
0693 08                INX
0694 DF 56             STX    KADR
0696 DE 5A             LDX    SADR      Increment table pointer to next table.
```

```
600  0698 C6 20        LDA B #32
601  069A BD AD 36     JSR   ADDBS
602  069D DF 5A        STX   SADR
603  069F 33           PUL B             Get iteration counter.
604  06A0 5A           DEC B
605  06A1 2E 9C        BGT   PERMS1      Keep going.
606  06A3 CE 00 2F     LDX   #K
607  06A6 DF 56        STX   KADR        Reset source data pointer.
608  06A8 CE 00 37     LDX   #K1
609  06AB DF 58        STX   K1ADR       Reset result data pointer.
610  06AD CE 01 BE     LDX   #S
611  06B0 DF 5A        STX   SADR        Reset table pointer.
612  06B2 39           RTS
613                    END

NO ERROR(S) DETECTED

   SYMBOL TABLE:

ADDBS  0036   ADJ    03C0   AINPUT 001C   APARM  0044   AMESGD 0020
BADDR  ED47   BFFADR 0469   BUFFER 041D   BYTE   E095   C      004D
CIPHER 008E   CLOSE  A03F   CPTXTM 04B3   D      004F   DECODE 03FE
DECP   0579   E      01AE   ENCODE 0373   ENT    0003   GETCHR AD15
GETCPH 03DA   GETFIL AD2D   GETL   0662   GETPLN 0336   GOON   0593
GOTCPH 03F5   GOTPLN 0365   INEEE  E1AC   INKEY  0123   INPUT  0043
INPUT1 0060   INPUT2 0075   INTTAB 00C6   INVTAB 020E   ITIR   053F
ITER1  0554   JFLAG  0066   K      002F   K1     0037   K1ADR  0058
KADR   0056   KEY    005C   KEY1   007E   KEY2   0086   KEYMSG 0006
KPARM  00A6   L      0027   LOOP1  04ED   LSHIFT 0310   MADR1  0068
MADR2  006A   MAIN   02FE   MODE   04D4   MOVE   04E0   NBT1   059A
NS 2   0586   NBIT   0024   NBUF   046F   NOADJ  03E2   NPIECE 001D
NRESET 001C   NSHFL  0600   NSHFR  067A   NSTR   0504   NUMER  03C3
NUMER1 03E0   OUT2H  E0BF   OUT2HS E0CA   OUT4HS E0C8   OUTEEE E1D1
OUTS   E0CC   OUTSHF 053C   P      019E   P1     05E2   PARMS  008E
PCH1A  0100   PCH1B  0122   PCH2   013E   PCPHSG 0399   PCRLF  AD24
PDATA1 E07E   PERM   05DB   PERMS  063D   PERMS1 063F   PLAIN  0067
PLOOP  0593   PSIDE  046D   PSTRNG AD1E   PTXTHS 0493   PUTCHR AD18
R      002D   RB     0025   REDND  003D   ROT1   04DA   ROTATE 04D5
RPIECE A03F   RSHIFT 0525   S      01BE   SADR   0034   SAV    0070
SBIT   0053   SETEXT AD33   SHFIN  03FC   SHIFT  0303   SLOOP  0612
SNUM   0054   STL    0685   STLOC  001A   STO    0699   SVI    05C9
TABADR 0022   TRASH  0021   UPRBG  0070   VM     0002   WARMS  A003
```

```
   EXAMPLE 1
***DESSERT
TRANSMITTER OR RECIEVER (00 OR 01)
00
INPUT KEY (8 BYTES)
1234567890ABCDEF
INPUT PLAIN TEXT FOLLOWED BY CR
SYNCWORD THIS IS PLAINTEXT !
28 B0 AD C4 F0 F1 FB E0 78 F4 90 71 0E A0 F6 ED 2F C5 6D 64 28 9A A5 CA BE 3D 01 47 13
***
```

```
   EXAMPLE 2
***DESSERT
TRANSMITTER OR RECIEVER (00 OR 01)
01
INPUT KEY (8 BYTES)
1234567890ABCDEF
INPUT CIPHER TEXT FOLLOWED BY CR
28 D0 AD C4 F0 F1 FB ED 78 F4 90 71 0E A0 F6 E3 2F C5 6B 64 28 9A A5 CA BE 30 01 47 13
*IOP THIS IS PLAINTEXT !
***
```

```
   EXAMPLE 3
***DESSERT
TRANSMITTER OR RECIEVER (00 OR 01)
01
INPUT KEY (8 BYTES)
2234567890ABCDEF
INPUT CIPHER TEXT FOLLOWED BY CR
28 D0 AD C4 F0 F1 F3 EB 78 F4 90 71 0E A0 F6 EB 2F C5 6D 64 28 9A A5 CA BE 3D 01 47 13
\ {zT'f2+0dq0-t(gmg]0e8d
***
```

```
   EXAMPLE 4
***DESSERT
TRANSMITTER OR RECIEVER (00 OR 01)
01
INPUT KEY (8 BYTES)
1234567890ABCDEF
INPUT CIPHER TEXT FOLLOWED BY CR
28 D0 AD C4 F0 F1 FD EB 78 F4 90 70 0E A0 FA ED 2F C5 4D 64 28 9A A5 CA BE 3D 01 47 13
0NRP7THEXT !
***
   EXAMPLE 5
***DESSERT
TRANSMITTER OR RECIEVER (00 OR 01)
00
INPUT KEY (8 BYTES)
1234567887654321
INPUT PLAIN TEXT FOLLOWED BY CR
Syncword This shows lower case!
11 C6 FE 67 B5 AF F6 B0 D3 C6 8A 17 22 77 07 A2 8F 05 05 B2 4C 0A 0E 3F 24 4B FC EC 12 D8 14 75
***
```

```
   EXAMPLE 6
***DESSERT
TRANSMITTER OR RECIEVER (00 OR 01)
01
INPUT KEY (8 BYTES)
1234567887654321
INPUT CIPHER TEXT FOLLOWED BY CR
11 C6 FE 67 B5 AF F4 B0 D3 C6 8A 17 22 77 07 A2 8F 05 05 B2 4C 0A 0E 3F 24 4B FC EC 12 D8 14 75
Qn'0 This shows lower case!
```

```
   EXAMPLE 7
***DESSERT
TRANSMITTER OR RECIEVER (00 OR 01)
00
INPUT KEY (8 BYTES)
1234567887654321
INPUT PLAIN TEXT FOLLOWED BY CR
CODESYNC CHARACTERS:<>?*!@#$..
01 FB EC EE 30 95 4E E9 F1 25 7E 75 3D C5 E2 07 7F 47 39 B7 8A 0A B2 F2 E5 E9 AE E7 34 F4
***
```

```
   EXAMPLE 8
***DESSERT
TRANSMITTER OR RECIEVER (00 OR 01)
01
INPUT KEY (8 BYTES)
1234567887654321
INPUT CIPHER TEXT FOLLOWED BY CR
01 FB EC EE 30 95 4E E9 F1 25 7E 75 3D C5 E2 07 7F 47 39 B7 8A 0A B2 F2 E5 E9 AE E7 34 F4
AunFu900 CHARACTERS:<>?*!@#$..
***
```

BOOKEEPING (Part-3)
Final next month.

William Stock
1125 Lois Dr.
Cincinnati, OH 45237

```
0001 REM PROFIT/LOSS
0002 REM PRINTS INCOME & EXPENSE
0003 REM AND CALCULATES DIFFERENCE
0004 REM   PL.BAS
0010 OPEN #1,0.PLM
0020 READ #1,V1,V2,V3,V4,V5,V6,V7,V8,V9
0030 CLOSE #1
0040 OPEN #1,1.GLMASTER
0050 PRINT CHR$(16);CHR$(22);CHR$(0);CHR$(0);CHR$(0);
0060 REM FOR PRINTED OUTPUT INSERT
0070 REM PORT INFORMATION HERE
0500 PRINT "PROFIT/LOSS AS OF ";
0510  A$=STR$(V1)
0520 IF V1<99999 THEN A$="0"+A$
0530 PRINT LEFT$(A$,2);"/";MID$(A$,3,2);"/";RIGHT$(A$,2)
0540 PRINT
0550 PRINT "INCOME"
0560 PRINT
0570 GOSUB 1000:REM READ FILE
0580 IF G1>V4 THEN 630
0585 IF G2=0 THEN 570
0590 GOSUB 1100:REM DECIMAL ALIGN
0600 PRINT G1;G$;TAB(32-LEN(A$));A$
0610  P=P+G2
0620 GOTO 570
0630 PRINT
0640  G2=P:GOSUB 1100
0650 PRINT "TOTAL INCOME";TAB(25-LEN(A$));A$
0660 READ #1,G1,G2,G$:IF EOF(1)=1 THEN PRINT "FILE INCOMPLETE":GOTO 850
0670 IF G1<=V6 THEN 660
0680 PRINT :PRINT:PRINT "EXPENSES":PRINT
0690 GOTO 710
0700 GOSUB 1000:REM READ FILE
0710 IF G1>V7 THEN 760
0715 IF G2=0 THEN 700
0720 GOSUB 1100:REM DECIMAL ALIGN
0730 PRINT G1;G$;TAB(32-LEN(A$));A$
0740  L=L-G2
0750 GOTO 700
0760 PRINT
0770  G2=L:G1=0
0780 GOSUB 1100:REM DECIMAL ALIGN
0790 PRINT "TOTAL EXPENSES";TAB(25-LEN(A$));A$
0800 PRINT
0810  G2=P-L
0820 GOSUB 1100
0830 IF L<P THEN PRINT "PROFIT";TAB(25-LEN(A$));A$:GOTO 850
0840 PRINT "LOSS!";TAB(25-LEN(A$));A$
0850 CLOSE #1
0860 PRINT
0870 INPUT A$
0880 CHAIN 0.MENU
0993 REM
0995 REM READ SUBROUTINE
0997 REM
1000 READ #1,G1,G2,G$
1010 IF EOF(1)=1 THEN G1=99999999
1020 RETURN
1093 REM
1095 REM DECIMAL ALIGN
1097 REM
1100  A$=STR$(G2)
1105 IF G1>V6 THEN A$=STR$(-G2)
1110 IF G2=INT(G2) THEN A$=A$+".0"
1120 IF G2*10=INT(G2*10) THEN A$=A$+"0"
1130 RETURN
```

```
0001 REM MAINTENANCE ROUTINE
0002 REM BOTH G/L & A/P.
0003 REM    MAIN.BAS
0005 STOP :END
0010  F=2:GOTO 1000:REM A/P MAINT
0020  F=1:GOTO 1000:REM G/L MAINT
0093 REM
0095 REM SUBROUTINES
0097 REM
0100 PRINT CHR$(16);CHR$(22);CHR$(0);CHR$(0);CHR$(0);
0110 RETURN
0120 GOSUB 100
0130 FOR X=1 TO 30
0140 PRINT "*";
0150 NEXT X
0155 PRINT :PRINT
0160 PRINT "MAINTENANCE IS FATHER/SON"
0170 PRINT "ACCT #'S MUST BE IN"
0180 PRINT "ASCENDING SEQUENCE."
0190 PRINT :PRINT
0200 FOR X=1 TO 30
0210 PRINT "*";
0220 NEXT X
0222 OPEN #1,0.PRM
0225 READ #1,V1,V2,V3,V4.V5,V6.V7,V8,V9
0227 CLOSE #1
0230 PRINT :PRINT
0240 RETURN
0243 REM
0245 REM G/L OPEN
0250 INPUT "IS OLDEST G/L ON #0",A$
0255 IF A$="" THEN 250
0260 IF LEFT$(A$,1)<>"Y" THEN PRINT "PUT IT ON":GOTO 250
0270 OPEN #1,1.GLMASTER
0280 OPEN #2,0.GLMASTER.DAT
0290 SCRATCH #2
0300 RETURN
0303 REM
0305 REM A/P OPEN
0307 REM
0310 INPUT "IS OLDEST A/P ON #0",A$
0315 IF A$="" THEN 310
0320 IF LEFT$(A$,1)<>"Y" THEN PRINT "PUT IT ON":GOTO 310
0330 OPEN #1,1.APMASTER
0340 OPEN #2,0.APMASTER.DAT
0350 SCRATCH #2
0360 RETURN
0400 REM
0410 REM WRITE RECORD
0415 REM
0420 WRITE #2,O1,O2,O$,O3,O4
0430 RETURN
0433 REM
0435 READ RECORD
0437 REM
0440 READ #1,I1,I2,I$,I3,I4
0450 NK)BDDBRB-% EOF(1)=1 THEN I1=99999999
0460 RETURN
0470 REM
0480 REM
0500 IF VAL(A$)<=V7 THEN V=1
0510 IF VAL(A$)>V8 THEN V=1
0520 IF VAL(A$)<O1 THEN V=1
0530 RETURN
0540 REM
0550 REM
0993 REM
0995 REM MAIN PROGRAM STARTS HERE
0997 REM
1000 GOSUB 120
1010 ON F GOSUB 250,310:REM A/P,G/L
1020 GOSUB 440
1030 GOSUB 100
1040 INPUT "ACCT #",A$
1050 IF A$="" THEN 1500:REM CLOSE UP
1060  V=0
1070 ON F GOSUB 520,500:REM VALIDATE ACCT #
1080 IF V=1 THEN INPUT "INVALID ACCT.      ",A$:GOTO 1030
1090  A=VAL(A$)
1100 IF A>I1 THEN 1430:REM FIND ACCT
1110 IF A=I1 THEN 1300:REM THIS IS IT
1120 PRINT "NEW ACCOUNT"
1130  O1=A:O2=0
1140 INPUT "DESCRIPTION",O$
1150 IF F=1 THEN 1220
1160 INPUT "PAYMENT $ ",O3
1170  O3=-ABS(O3)
1190 INPUT "PMT DUE (MM:DD - DD)",A$
1200 IF A$="" THEN A$="0"
1210  O4=VAL(A$)
1220 INPUT "DATA OK",A$
1225 IF A$="" THEN 1220
1230 IF LEFT$(A$,1)<>"Y" THEN 1140
1240 GOSUB 420:REM WRITE RECORD
1250 GOTO 1030
1300  O1=I1:O2=I2:O3=I3:O4=I4:O$=I$
1310 PRINT "OLD DESCR: ";I$
1320 INPUT "NEW DESCR:",O$
1323 IF O$="" THEN O$=I$
1325 IF O$="DELETE" THEN IF I2=0 THEN 1410
1330 IF F=1 THEN 1390
1340 PRINT "OLD PAYMENT: ";ABS(I3)
1350 INPUT "NEW PAYMENT",O3:O3=-ABS(O3)
1360 PRINT "OLD DATE: ";I4
1370 INPUT "NEW DATE ",O4
1380 INPUT "DATA OK",A$
1385 IF A$="" THEN 1380
1390 IF LEFT$(A$,1)<>"Y" THEN GOSUB 100:GOTO 1300
1400 GOSUB 420: REM WRITE RECORD
1410 GOSUB 440:REM READ NEXT
1420 GOTO 1030
1423 REM
1425 REM FIND ACCT
1427 REM
1430 IF EOF(1)=1 THEN 1500
1440  O1=I1:O2=I2:O3=I3:O4=I4:O$=I$
1450 GOSUB 420:REM WRITE ACCT
1460 GOSUB 440
1470 GOTO 1100
1493 REM
1495 REM COPY REST OF FILE
1497 REM
1500 IF EOF(1)=1 THEN 1550:REM COPY DONE, FINNISH
1510  O1=I1:O2=I2:O3=I3:O4=I4:O$=I$
1520 GOSUB 420:REM WRITE RECORD
1530 GOSUB 440:REM READ NEXT
1540 GOTO 1500
1550 IF EOF(1)=1 THEN 1550:REM COPY DONE - FINISH
1560 GOSUB 100
1570 IF F=2 THEN 1670:REM NO HISTORY FOR A/P
1580 PRINT "COPYING DETAIL"
1590 OPEN #1,1.GLHIST
1600 OPEN #2,0.GLHIST.DAT
1610 SCRATCH #2
1620 READ #1,I1,I2,I3,I4
1630 IF EOF(1)=1 THEN 1660
1640 WRITE #2,I1,I2,I3,I4
1650 GOTO 1620
1660 CLOSE #1,#2
1670 PRINT "NEW MASTER ON #0."
1675 PRINT "PUT SYSTEM DISC ON #0."
1680 INPUT "IS IT THERE",A$
1685 IF A$="" THEN 1680
1690 IF LEFT$(A$,1)<>"Y" THEN 1675
1700 CHAIN 0.MENU
```

```
0001 REM RECOVERS G/L AND JOURNAL
0002 REM FROM TAPE
0003 REM REQUIRES SUIBUG, BASIC 3.0, & MINIFLEX 1.0
0004 REM    RECOV.BAS
0010 GOTO 1000
0093 REM
0095 REM SUBROUTINES
0100 PRINT CHR$(16);CHR$(22);CHR$(0);CHR$(0);CHR$(0);
0110 RETURN
0193 REM
0195 REM READ TAPE, WRITE DISC
0197 REM
0200 OPEN #1,B$
0210 SCRATCH #1
0215 WRITE #1,A:REM LET BASIC DO WORK
0220 POKE( 103,361
0230 IF B$="1.JOURNAL.DAT" THEN POKE(104,75):GOTO 250
0240 IF B$="1.GLHIST.DAT" THEN POKE(104,78):GOTO 250
0245 POKE( 104,81)
0250  A=USER(X)
0260 IF A=0 THEN 360
0265 IF A=B THEN IF T=1 THEN 290
0270 IF A=B THEN 290
0280 PRINT "UNSUCCESSFUL RECOVERY":GOTO 310
0290 PRINT "WRONG TAPE"
0300 PRINT "I NEED ";B$
0310 INPUT "DO YOU WANT TO TRY AGAIN",A$
0320 IF A$="" THEN 310
0330 IF LEFT$(A$,1)="Y" THEN T=1:POKE(104,81):GOTO 250
0340 IF LEFT$(A$,1)="N" THEN RETURN
0350 GOTO 310
0360 PRINT B$;" RECOVERED"
0370 RETURN
0993 REM
0995 REM PROGRAM STARTS HERE
0997 REM
1000 GOSUB 100
1010 PRINT "RECOVER GENERAL LEDGER (G)"
1020 PRINT TAB(9);"JOURNAL";TAB(24);"(J)"
1030 PRINT TAB(6);"OR NEITHER";TAB(24);"(N)"
1040 INPUT A$
1050 IF A$="" THEN 1000
1060 IF LEFT$(A$,1)="G" THEN 1200
1070 IF LEFT$(A$,1)="J" THEN 1400
1080 IF LEFT$(A$,1)="N" THEN CHAIN 0.MENU
1090 PRINT "G, J, OR N"
1100 GOTO 1040
1193 REM
1195 REM G/L RECOVERY
1197 REM
1200 GOSUB 100
1210 PRINT "I NEED OLDEST G/L ON #1"
```

```
1220 INPUT "IS IT THERE",A$
1230 IF A$="" THEN 1220
1240 IF LEFT$(A$,1)<>"Y" THEN 1200
1250  B$="1.GLHIST.DAT"
1260 GOSUB 200
1270 CLOSE #1
1280  B$="1.GLMASTER,DAT"
1290 GOSUB 200
1300 CLOSE #1
1310 CHAIN O.MENU
1393 REM
1395 REM JOURNAL RECOVERY
1397 REM
1400 GOSUB 100
1410 PRINT "I NEED OLDEST JOURNAL ON #1"
1420 INPUT "IS IT THERE",A$
1430 IF A$="" THEN 1420
1440 IF LEFT$(A$,1)<>"Y" THEN 1400
1450  B$="1.JOURNAL.DAT"
1460 GOSUB 200
1470 CLOSE #1
1480 CHAIN O.MENU
```

```
0001 REM CHANGES SEQ # BACK TO 1
0002 REM DURING END OF YEAR PROCEDURE
0003 REM    CHNG.BAS
0010 OPEN #1,0.PRM
0020 READ #1,V1,V2,V3,V4,V5,V6,V7,V8,V9
0025 CLOSE #1:OPEN #1,0.PRM
0030 SCRATCH #1
0040  V2=1
0050 WRITE #1,V1,V2,V3,V4,V5,V6,V7,V8,V9
0060 CLOSE #1
0070 CHAIN O.MENU
```

```
0001 REM BUILD PARAMETER FILE
0002 REM  INSTALL.BAS
0010 GOSUB 1000
0020 PRINT "PUT NEW SYSTEM DISC IN #0"
0030 INPUT "IS IT IN",A$
0040 IF LEFT$(A$,1)<>"Y" THEN 20
0050 OPEN #1,0.PRM.DAT
0060 PRINT "WHAT IS THE HIGHEST ACCT # FOR:"
0070 PRINT
0080 INPUT "INCOME ACCTS",V4
0090 INPUT "ACCTS RECEIVABLE",V5
0100 INPUT "ASSET ACCTS",V6
0110 INPUT "EXPENSE ACCTS",V7
0120 INPUT "ACCTS PAYABLE",V8
0130 INPUT "NET WORTH",V9
0140 GOSUB 1000
0150 PRINT V4;TAB(10);"INCOME"
0160 PRINT V5;TAB(10);"RECEIVABLES"
0170 PRINT V6;TAB(10);"ASSETS"
0180 PRINT V7;TAB(10);"EXPENSE"
0190 PRINT V8;TAB(10);"PAYABLE"
0200 PRINT V9;TAB(10);"NET WORTH"
0210 PRINT
0220 IF V4<V5 THEN IF V5<V6 THEN IF V6<V7 THEN 240
0230 GOTO 250
0240 IF V7<V8 THEN IF V8<V9 THEN 320
0250 PRINT "INCOME MUST BE LESS THAN"
0260 PRINT TAB(3);"RECEIVABLES"
0270 PRINT TAB(3);"WHICH MUST BE < ASSETS"
0280 PRINT TAB(3);"ETC., ETC., ETC."
0290 INPUT "PRESS RETURN WHEN READY.    ",A$
0300 GOSUB 1000
0310 GOTO 60
0320 INPUT "IS DATA CORRECT",A$
0330 IF LEFT$(A$,1)<>"Y" THEN 300
0340  V1=79218:V2=1:V3=0
0350 WRITE #1,V1,V2,V3,V4,V5,V6,V7,V8,V9
0360 CLOSE #1
0370 OPEN #1,0.PTRAN
0380 WRITE #1,A
0390 CLOSE #1
0400 GOSUB 1000
0410 PRINT "YOUR DISCS ARE BUILT"
0420 PRINT
0430 INPUT "HIT RETURN.     ",A$
0440 CHAIN O.APB
1000 PRINT CHR$(16);CHR$(22);CHR$(0);CHR$(0);CHR$(0);
1010 RETURN
```

```
0001 REM BUILD A/P & G/L MASTER FILES
0002 REM  APB-BAS
0003 REM HAVE ACCT #'S & BALANCES
0004 REM READY
0005 REM
0010 GOSUB 900
0020 GOSUB 800
0030 PRINT "PLACE G/L DISC ON #0"
0040 PRINT TAB(7);"A/P DISC ON #1"
0050 INPUT "ARE THEY THERE",A$
0060 IF LEFT$(A$,1)<>"Y" THEN 20
0070 PRINT "WE WILL NOW BUILD THE"
0080 PRINT "A/P MASTER FILES."
0090 PRINT :PRINT "ENTER THE INFORMATION"
0100 PRINT "REQUESTED.  WHEN FINISHED,"
0110 PRINT "PRESS RETURN FOR ACCT #."
0120 INPUT "READY",A$
0130 IF LEFT$(A$,1)<>"Y" THEN 120
0135 GOSUB 930:REM OPEN FILE
0140 GOSUB 1000:REM INPUT DATA
0150 IF P1=0 THEN 180
0155 IF P1<=B THEN INPUT "ASCENDING SEQUENCE",A$:GOTO 140
0160 WRITE #1,P1,P2,P$,P3,P4
0165  B=P1
0170 GOTO 140
0180 GOSUB 800
0190 INPUT "FINISHED WITH PAYABLES",A$
0200 IF LEFT$(A$,1)="Y" THEN 250
0210 IF LEFT$(A$,1)="N" THEN 140
0220 GOTO 180
0250 CLOSE #1
0260 KILL 0.GLMASTER.DAT
0270 OPEN #1,0.GLMASTER
0280 GOSUB 800
0290 PRINT "WE WILL NOW BUILD THE"
0300 PRINT "G/L MASTER FILE."
0310 PRINT :PRINT "ENTER THE INFORMATION"
0320 PRINT "REQUESTED.  WHEN FINISHED,"
0330 PRINT "PRESS RETURN FOR ACCT #."
0340 INPUT "READY",A$
0350 IF LEFT$(A$,1)<>"Y" THEN 340
0355  B=0
0360 GOSUB 1400:REM INPUT DATA
0370 IF G1=0 THEN 410
0375 IF G1<=B THEN INPUT "ASCENDING SEQUENCE!    ",A$:GOTO 360
0380 WRITE #1,G1,G2,G$
0383  B=P1
0385 IF G1<=V4 THEN P=P+G2:GOTO 360
0390 IF G1>V6 THEN IF G1<=V7 THEN P=P+G2:GOTO 360
0395  P=P-G2
0400 GOTO 360
0410 GOSUB 800
0420 INPUT "FINISHED WITH GEN LEDGER",A$
0425 IF A$="" THEN 420
0430 IF LEFT$(A$,1)="Y" THEN 500
0440 IF LEFT$(A$,1)="N" THEN 360
0450 GOTO 420
0500 OPEN #2,APMASTER
0510 READ #2,G1,G2,G$
0520 IF EOF(2)=1 THEN 560
0530 WRITE #1,G1,G2,G$
0540  P=P-G2
0550 GOTO 510
0560  G$="NET WORTH"
0570 WRITE #1,V9,P,G$
0580 CLOSE #1,#2
0590 GOSUB 800
0600 PRINT "YOUR CONVERSION IS FINISHED."
0610 PRINT :PRINT V9;G$;"  $";-P
0620 PRINT :INPUT "IS SYSTEM DISC IN #0",A$
0630 IF A$="" THEN 620
0640 IF LEFT$(A$,1)<>"Y" THEN 620
0650 CHAIN 0.START
0793 REM
0795 REM CLEAR SCREEN
0797 REM
0800 PRINT CHR$(16);CHR$(22);CHR$(0);CHR$(0);CHR$(0);
0810 RETURN
0893 REM
0895 REM OPEN A/P
0897 REM
0900 OPEN #1,0.PRM
0910 READ #1,V1,V2,V3,V4,V5,V6,V7,V8,V9
0920 CLOSE #1 :RETURN
0930 KILL 1.APMASTER.DAT
0940 OPEN #1,1.APMASTER
0950 RETURN
0993 REM
0995 REM A/P INPUT
0997 REM
1000 GOSUB 800
1010 PRINT TAB(20);"PAYABLES"
1020 PRINT :INPUT "ACCT #",P$
1030 IF P$="" THEN P1=0:RETURN
1040 IF ASC(P$)>57 THEN 1250
1050  P1=VAL(P$)
1060 IF P1>V8 THEN 1270
1070 IF P1<=V6 THEN 1270
1075 INPUT "DESCRIPTION",P$
1080 INPUT "BALANCE",P2
1090  P2=-P2
1110 INPUT "PAYMENT $",A$
1120 IF A$="" THEN P3=0:GOTO 1140
1130  P3=-VAL(A$)
1140 INPUT "PMT DUE DATE (MMDD OR DD)",A$
1150 IF A$="" THEN P4=0:GOTO 1170
1160  P4=VAL(A$)
1170 PRINT
1180 INPUT "IS DATA CORRECT",A$
1190 IF A$="" THEN RETURN
1200 IF LEFT$(A$,1)="Y" THEN RETURN
1210 PRINT "ENTRY REJECTED."
1220 INPUT "RETURN TO CONTINUE.    ",A$
1230 GOTO 1000
1250 INPUT "NUMBERS ONLY, OK",A$
1260 GOTO 1000
```

```
1270 PRINT "PAYABLES ONLY"
1280 GOTO 1020
1393 REM
1395 REM G/L INPUT
1397 REM
1400 GOSUB 800
1410 PRINT TAB(14);"GENERAL LEDGER"
1420 PRINT :INPUT "ACCT #",G$
1430 IF G$="" THEN G1=0:RETURN
1440 IF ASC(G$)>57 THEN 1680
1450  G1=VAL(G$)
1470 IF G1>V7 THEN 1620
1475 INPUT "DESCRIPTION",G$
1480 INPUT "BALANCE",G2
1485 IF G1>V6 THEN G2=-G2
1500 PRINT :INPUT "IS DATA CORRECT",A$
1510 IF A$="" THEN RETURN
1520 IF LEFT$(A$,1)="Y" THEN RETURN
1530 PRINT "ENTRY REJECTED."
1540 INPUT "RETURN TO CONTINUE.     ",A$
1550 GOTO 1400
1620 PRINT "A/P ALREADY DONE"
1630 IF G1=V9 THEN PRINT "NET WORTH IS BEING FIGURED."
1640 INPUT "OK",A$
1650 GOTO 1400
1680 PRINT "NUMBERS PLEASE!"
1690 GOTO 1540
```

DISK MODS

David Kyllingstad
840 Hillview Dr.
Marion, Iowa 52302

Enclosed are a couple of items that may be of interest. The first is a modification to minifloppy drives that inhibits the motors from running if the door is open. The other is yet another change to the mini drive bootstrap. THIS ONE WORKS!

Since the motor control line to the floppy drives is not a multiplexed line, all drives start and stop in unison. This causes undue wear and tear on drives not actually needed. The modification is quite simple and should be clear from the diagram. What is happening is that the base of Q1 is wire ORed to a new switch that is grounded when the door is in the full open position. The current is very low so nothing elaborate is required. Purists may want to use a micro switch but will end up fabricating a mounting bracket that is harder to make than the clip itself. The goal was to require the absolute minimum modification to the drive. Soldering one wire and pushing on one clip seems to meet that goal. The clip I used was a piece of beryllium copper channel that (in its former life) was a circuit card edge guide. Any thin metal such as hobby store brass stock that can be bent with ease and will accept solder will do. The piece I used was held on by its own spring tension. If soft brass is used, a small spot of glue may be required.

The next paragraph should demystify (except for why SWTPc did it that way) the interaction of the controller and the bootstrap loader. To boot the operating system from a MF-68 compatible disk, the first two sectors on track zero which are not IBM format must be loaded into memory at $2400 then control is transferred to $2400. Much of the difficulty people have been having booting their systems can be traced to two things. Early 1771s had trouble reading non IBM format data and would produce a CRC error indication even though the data had been read correctly and the loss of drive select when the controller unloads the head.

For those using the boot published in the FLEX (TM) manual which contains the instructions LDA B COMREG AND B $0C BNE START, the symptoms of a bad 1771 are a continuous cycling of the drive. Restore - load head - restore - load head.... At this point, the user may hit reset and jump to $2400. Everything will continue as normal but will hardly impress one's friends or relatives. If you have one of these chips you have two choices, trade it off to someone using entirely IBM format disks or take out the instructions mentioned above. SWTPc chose to do the latter as the boot in SWTBUG (TM) does not do the test. You should not encounter any problems reading any other sectors on the disk as they are IBM format (kind of).

The second problem is a result of the interaction of the 1771 and Shugart SA-400 drives. The SA-400 does not have a separate head load signal line. The user is instructed (by SWTPc) to strap head load to occur with select. The 1771 however, has a separate head load function. This signal is used to enable the decoding of drive selection (IC5) on the controller card. No head load, no drive is selected. This apparently created a problem for the bootstrap loader contained in SWTBUG (TM) as the other half of decoder IC5 was used to force the selection of drive zero as a result of a hardware reset. The 1771 issues a restore on reset so everything seemed okay and in fact, did work. There are two problems with the fix, however. One is that it is not a good practice to load the head on a stationary disk as scuffing may take place. The other is that I have trouble explaining to my kids at home and peers at work that you should not remove or insert a disk when the head load light is on which, of course, is true whenever the time delay has run out or a reset was necessary. The solution was to break the circuit between IC5 pins

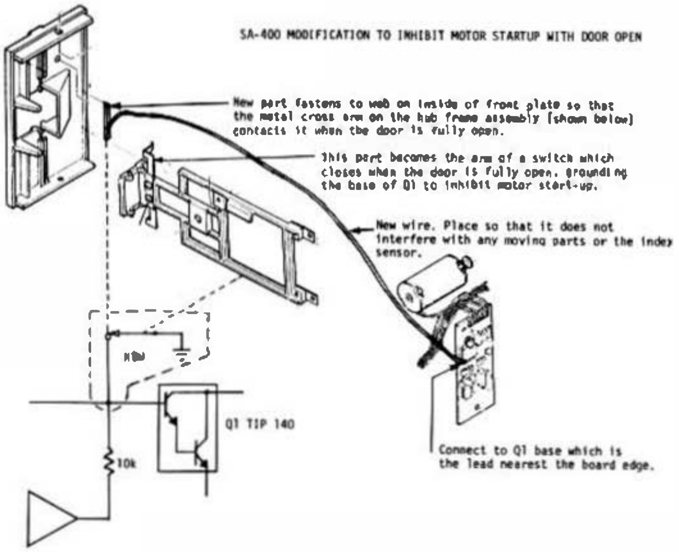
6 and 10. This was recommended by SWTPc in the form of an addendum (maybe only for those receiving WANGCO drives) to the assembly instructions and was the subject of a previous article in 68 MICRO JOURNAL. Now, existing bootstraps either won't work or there is a delay of 5 seconds after a reset.

The bootstrap program enclosed, addresses all the problems discussed and one other. It is in use on five systems and has not failed to work properly. Only a brief explanation should be necessary in addition to the comments in the program. You MUST clear the drive register to select drive Ø and start the motor. If bit 7 was latched as a 1 in IC8 as a result of power up junk, a program gone astray, inadvertent memory testing of I/O addresses, or whatever, the drive motors will not start - ever! With no drive selected because the 1771 unloads the head on restore, the track zero indication to the 1771 is not present so it does the best it can. It issues 255 step (out) pulses at 2Øms/step (5.1sec.) and sets busy. During this time, it will ignore ALL OTHER commands except force interrupt. Under these conditions, the restore in SWTBUG (TM), in the FLEX manual, and in the previously published boot, does not take place, so the boot reads from the current track! Hence, a fixed wait must be long enough to handle the worst case (the old "hold the door open for five Mississippis" trick), loops must test busy, as well as, ready (bits 1 and 7), or the hardware restore may be aborted with a force interrupt. I chose the last two. From here on, the program is unchanged except for the additional delay subroutines. This was my first attempt to fix the problem (without fully understanding it) and the TSC drivers had the additional delay, so it seemed like a good idea.

For those that have not seen previous articles on the subject, breaking the connection between IC4 pins 6 and 10 can be accomplished several ways. If you have a socket under IC4, remove IC4, bend pin 6 outward, and reinstall it in the socket. If you don't have a socket, you may either cut the pin 6 lead off where it enters the board, (use a sharp XACTO knife and great care then bend the lead out to clear all circuitry) or, you may remove IC4, cut the circuit foil between pins 6 and 10, and reinstall the IC.

One more thing, if a disk operation is attempted within 5.1 seconds of a reset, and the drive was not already at track zero, a DOS error 15 will result. If someone asks, change the subject.

Hope this clears up the puzzle for anyone experiencing difficulties.



SA-400 MODIFICATION TO INHIBIT MOTOR STARTUP WITH DOOR OPEN

New part fastens to web on inside of front plate so that the metal cross arm on the hub frame assembly (shown below) contacts it when the door is fully open.

This part becomes the arm of a switch which closes when the door is fully open, grounding the base of Q1 to inhibit motor start-up.

New wire. Place so that it does not interfere with any moving parts or the index sensor.

Connect to Q1 base which is the lead nearest the board edge.

```
BOOT                              TSC ASSEMBLER  PAGE 1     1/9/80 12 11

TSC ASSEMBLER VER 1.12
INPUT FILE IS BOOT TXT      1/7/80 ON DISK SCRATCH SYS,00 A    1/9/80
OUTPUT FILE IS BOOT BIN      1/9/80 ON DISK SCRATCH SYS,002A    1/9/80

                        NAM    BOOT
                  *
                  * IMPROVED BOOT THAT REQUIRES NO TRICKY
                  * DOOR MANIPULATION AND WORKS WITH CONTROLLER
                  * IC4 PIN 6-10 CONNECTION BROKEN. NO LONG DELAY
                  * IS ENCOUNTERED AFTER A RESET. DOES NOT DEPEND
                  * ON HARDWARE FOR DRIVE SELECTION OR TRACK ZERO
                  *
 8018             DISK  EQU    $8018      SWTPc PORT 6
 8014             DRVREG EQU   DISK-4
 8018             COMREG EQU   DISK
 801A             SECREG EQU   DISK+2
 801B             DATREG EQU   DISK+3
 2400             WHERE EQU    $2400      RAM LOAD LOCATION
                  *
 0100                   ORG    $0100
                  *
 0100 4F          MINI  CLR A             TO SELECT DRIVE ZERO AND
 0101 B7 80 14          STA A  DRVREG     INSURE MOTOR STARTUP
 0104 86 D0             LDA A  #$D0       CLEAR 1771 WITHOUT INTERRUPT
 0106 B7 80 18          STA A  COMREG     ABORTS HARDWARE RESTORE
 0109 F6 80 18    LOOP  LDA B  COMREG     START MOTOR AND GET 1771 STATUS
 010C C5 81             BIT B  #%10000001 TEST READY AND BUSY
 010E 26 F9             BNE    LOOP       WAIT
 0110 C6 0F             LDA B  #%00001111 RESTORE WITH HEAD LOAD
 0112 F7 80 18          STA B  COMREG
 0115 8D 31             BSR    DELAY1
 0117 F6 80 18    LOOP1 LDA B  COMREG
 011A C5 01             BIT B  #%00000001
 011C 26 F9             BNE    LOOP1      STILL BUSY
 011E 7F 80 1A          CLR    SECREG
 0121 8D 25             BSR    DELAY1
 0123 C6 9C             LDA B  #%10011100 READ MULTIPLE (IBM) RECORDS
 0125 F7 80 18          STA B  COMREG     WITH LONG HEAD LOAD DELAY
 0128 8D 1E             BSR    DELAY1
 012A CE 24 00          LDX    #WHERE
 012D C5 02       LOOP2 BIT B  #%00000010 DATA REQUEST
 012F 27 06             BEQ    LOOP3
 0131 B6 80 1B          LDA A  DATREG     LOAD A BYTE
 0134 A7 00             STA A  0,X        INTO MEMORY
 0136 08                INX
 0137 F6 80 18    LOOP3 LDA B  COMREG
 013A C5 01             BIT B  #%00000001 BUSY?
 013C 26 EF             BNE    LOOP2
 013E F6 80 18          LDA B  COMREG
 0141 C4 0C             AND B  #%00001100 IF CRC OR LOST DATA ERR
 0143 26 BB             BNE    MINI       GO AGAIN - ELSE
 0145 7E 24 00          JMP    WHERE      DO REMAINDER FROM DISK
 0148 8D 00       DELAY1 BSR   DELAY2
 014A 8D 00       DELAY2 BSR   DELAY3
 014C 39          DELAY3 RTS
                  *
                        END    MINI

NO ERROR(S) DETECTED
```

John Alford
10000 Midlothian Tpk.
Richmond, VA 23235
804-272-2005

THE CASE OF THE MISSING INTERRUPTS

(PATCHING SSB DOS68.51C ET AL.)

Those of us using SSB's DOS have one thing in common -- no interrupts allowed. Any program that uses interrupts will die the bad death when trying to use the DFM portion of DOS. The consistent answer from SSB has been 'but we know the interrupt status is put back!!'

I found that they were almost correct. Looking at the code in FIGURE ONE, they save the status in address $C4CB and put it back by or'ing it with the status returned by DFM. As with any good try at programming, though, they wrote code that looked good but didn't work. Apparently someone forgot that or'ing will not mask a logic ONE to a logic ZERO, and that is just what must be done since interrupts are enabled if the interrupt masking bit in the condition code register is zero.

By patching the code as listed in FIGURE TWO, the interrupt bit is properly handled by forcing a zero and then or'ing it with the saved status. Therefore, if the initial state of the interrupt bit was one it is restored to a one; if zero, it is returned to zero status. At last real time clock and queue-buffered I/O can be accomplished under SSB DOS!

The patches shown are for 5.1C DOS and a little digging may be required for other versions, but shouldn't be a great problem for most hackers to find. Our only regret is that we did not have a little more time to find the bug sooner.

```
DD69 07          TPA            GET STAT
DD6A 84 10       ANDA #$10      GET IBIT
DD6C B7 C4CB     STAA $C4CB     SAVE IT
DD6F 01          NOP            IN CASE
DD70 0F          SEI            MASK INT
DD71 AD 00       JSR  0,X       DO I/O
DD73 07          TPA            GET STAT
DD74 BA C4CB     ORAA $C4CB     HUH!!!
DD77 06          TAP            PUT STAT
```

FIGURE ONE - THE BUG IN DOS

```
SS69 07          TPA            GET STAT
DD6A 84 10       ANDA #$10      GET IBIT
DD6C B7 C4CB     STAA $C4CB     SAVE IT
DD6F 01          NOP            IN CASE
DD70 0F          SEI            MASK INT
DD71 AD 00       JSR  0,X       DO I/O
DD73 07          TPA            GET STAT
DD74 7E A04A     JMP PATCH      DO PATCH
DD77 06          TAP            PUT STAT
     .
A04A 84 EF       ANDA #$EF      MASK BIT
A04C BA C4CB     ORAA $C4CB     PUT BIT
A04F 7E DD77     JMP  $DD77     GET BACK
```

FIGURE TWO - THE PATCHED CODE

6800 and 6809 BOTH!!

Keith Alexander
681 Whitmore Rd., 207
Detroit, MI 48203

I recently wrote up an article on some simple hardware modifications which allow one to unconditionally use either a 6809 or 6800 processor board in the SWTPc computer system.

I applied it to my system this summer, when I got my '09 board, and found out that I wouldn't be able to simply switch back and forth between processors by just switching boards. As all SWTP '09 owners know, the supplied firmware, SBUG-E, addresses I/O at $E000, compared to $8000 in MIKBUG and SWTBUG. This problem is compounded by the "hard-wire" decoding on the motherboard, also placing I/O at $8000. Anyway, after writing up the 'fix' article and making sketches and so on, I realized I wasn't telling the whole story. I'd forgotten all about another mod I'd made earlier that really made the second one possible. That modification was toward 'tightening up' the I/O addressing of the SWTP motherboard, so these eight slots respond to exactly 32 unique memory addresses. As you know, the present circuitry doesn't fully define the I/O addresses, so they're spread out over 4K or 8K, depending on which motherboard you have.

Consequently, I realized that what was necessary was one rather long article or two shorter ones. The 'tightening' idea wasn't my own, just the method of implementation, and I give proper credit (I hope) in the actual text. It alone it worth doing, even if you aren't trying to relocate I/O. It frees up a lot of wasted memory space for the user. I first discovered a need for it when I started playing around with the addressing circuitry on SWTP 4K and 8K memory boards, trying to put them above $8000.

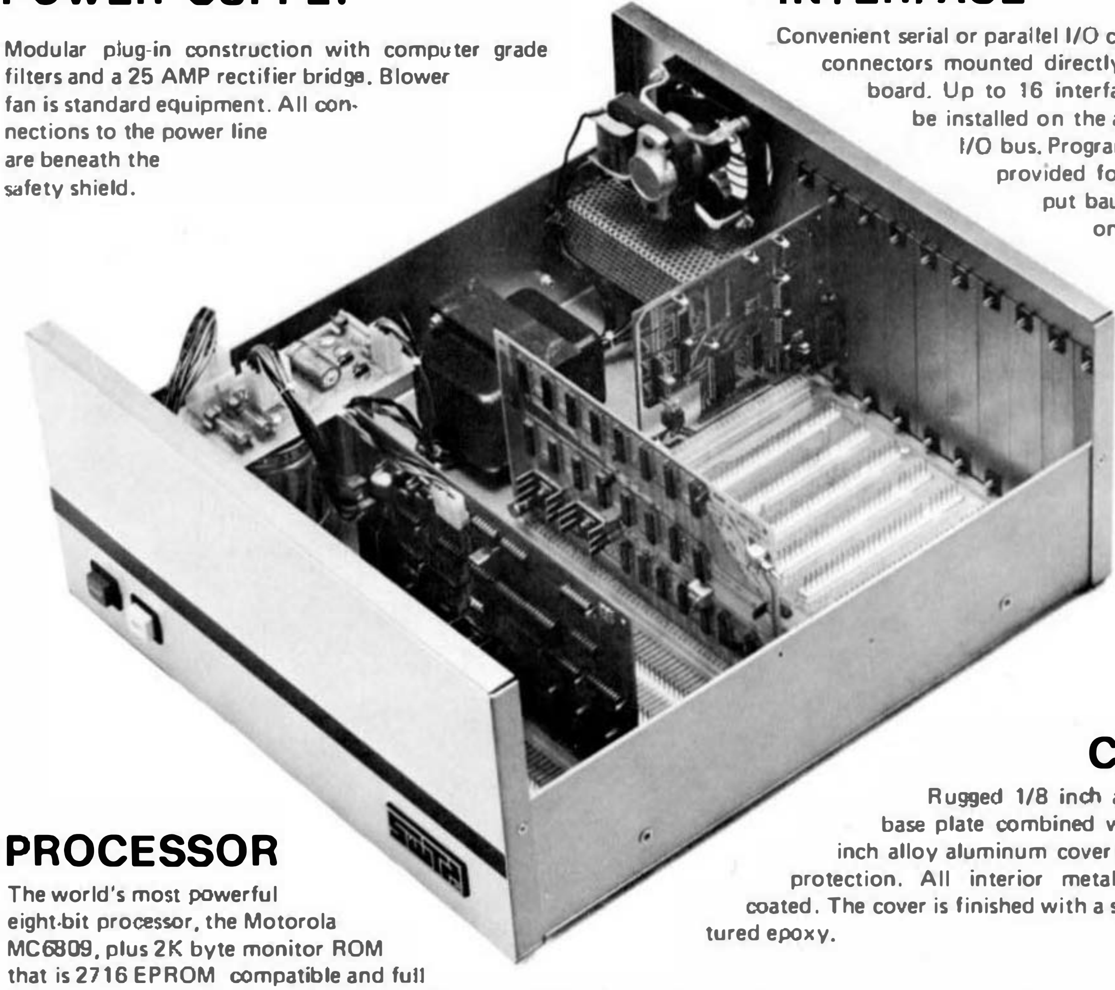
WE HAVE A 6809 FOR YOU

POWER SUPPLY

Modular plug-in construction with computer grade filters and a 25 AMP rectifier bridge. Blower fan is standard equipment. All connections to the power line are beneath the safety shield.

INTERFACE

Convenient serial or parallel I/O cards have DB-25 connectors mounted directly on the circuit board. Up to 16 interface devices may be installed on the address decoded I/O bus. Programming strips are provided for input and output baud rate selection on each port. All outputs are fully buffered.



CABINET

Rugged 1/8 inch alloy aluminum base plate combined with a solid 1/8 inch alloy aluminum cover for unsurpassed protection. All interior metal is conversion coated. The cover is finished with a super tough textured epoxy.

PROCESSOR

The world's most powerful eight-bit processor, the Motorola MC6809, plus 2K byte monitor ROM that is 2716 EPROM compatible and full buffering on all output lines. Built-in multiuser capability, just add I/O cards to operate a multi-terminal system.

MEMORY— You can purchase the computer with either 8K bytes of RAM memory (expandable to 56K), or with the full 56K. The efficient, cool running dynamic memory used in this system is designed and manufactured for us by "Motorola Memory Systems Inc."

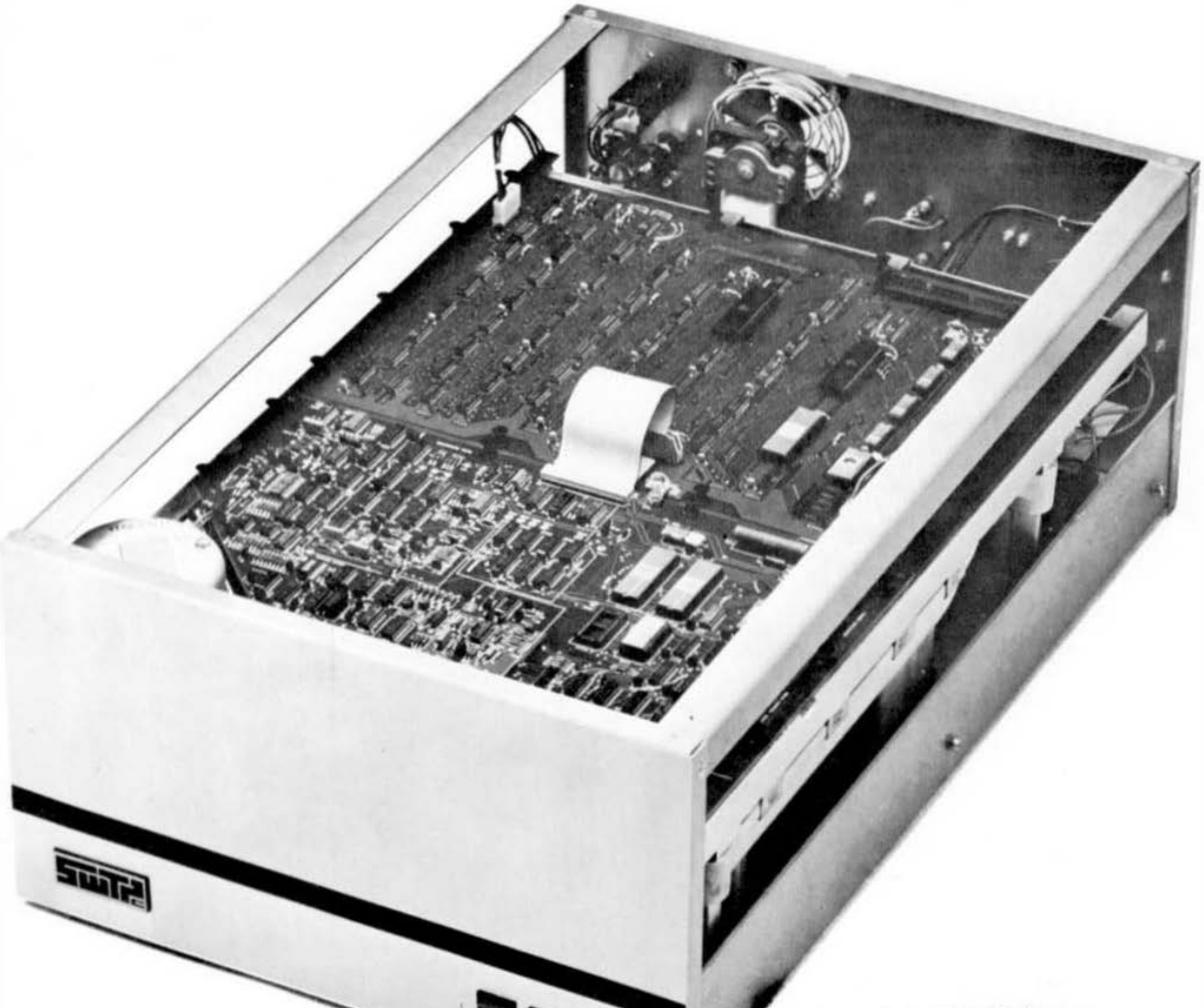
PERIPHERALS—The wide range of peripheral hardware that is supported by the 6809 includes: dot matrix printers (both 80 and 132 column), IBM Electronic 50 typewriter, daisy wheel printers, 5-inch floppy disk system, 8-inch floppy disk systems and a 16 megabyte hard disk.

SOFTWARE— The amount of software support available for the 6809 is incredible when you consider that it was first introduced in June, 1979. In addition to the FLEX9 operating system, we have a Text Editor, Mnemonic Assembler, Debug, Sort-Merge, BASIC, Extended BASIC, MultiUser BASIC, FORTRAN, PASCAL and PILOT.

69/K Computer Kit with 8K bytes of memory	**$ 495.00**
69/A Assembled Computer with 8K bytes of memory	**$ 595.00**
69/56 Assembled Computer with 56K bytes of memory	**$1,495.00**

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241



Think BIG

State of the art "Winchester"type hard disk with a data storage capacity of nearly 16 Megabytes, makes the SWTPC 6809 system the most flexible as well as the most powerful eight-bit microcomputer system in the world. The intelligent controller, using DMA data transfer, makes maximum use of the "Winchester" capability. It is completely compatible with the FLEX9 operating system used on the SWTPC 6809 floppy disk system.

CDS-1 "Winchester" disk drive with controller. .	$3,995.00
Cabinet–matching our 6809 computer desk. . .	150.00

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

Having just written a short article on how to relocate the I/O address space on the SWTP motherboards, I was reminded of a basic dilemma in this area, and how I have almost taken the solution I've implemented for granted. The dilemma is that the addressing of the I/O address space on the SWTP mother bus is not complete; that is, it's not specific enough, all bits are not defined. What happens is that each I/O slot doesn't occupy four unique addresses, but a range of groups-of-four-locations. When referring to the lowest address of I/O port 1, for example, you could just as easily refer to $9FC4 as $8004 (on the MP-B motherboard), or even $8784. They're all the same thing as far as the present decoder circuitry is concerned.

I found this "fix" in an article by Mr. Earle Hilton in INTERFACE AGE magazine, the Sept.'78 issue. The idea is to positively define address bits A6-A11 (or through A12) which are all 'don't care' bits presently. Whether the A12 bit is defined depends on which motherboard you have, -B or -B2.

My implementation required one foil cut on the bottom of the motherboard, attachment (by "piggyback" technique) of two cheap IC's, and some wire wrapping. And, of course, soldering. Which brings me to a few caveats. Use a small soldering iron, or a small tip. Hopefully, you will never have to solder anything smaller than 30-ga. wire-wrap to an IC pin. I've been thinking of hiring a highly-trained gang of fleas for just this sort of work, and under no circumstances RIDE in anything employing this (handy) construction technique. You should solder the wrapped connections to IC pins because they don't hold a wrap like a square post.

What you can do is lay down your added 74LS02 and 74LS30 (or LS20) right on top of two existing 14-pin ICs already on both motherboards, ICs 4 and 5. First you must carefully bend up all the pins except power and ground, usually 14 and 7 (or 16 and 8). Those bent up should be bent close to 180 degrees, while power and ground should be bent inward just enough to be sure they'll grip the corresponding pins of a similar IC they're "piggybacked" upon. Solder should carefully be applied to these two joints. You don't need to press the guest down into the host's socket or holes. Just a polite smooching, thank you, and a dab of heat and solder. Passing wires from the address bus to the ICs can be accomplished using several available feed-thrus in the area. I used three right in front of existing ICs 4 & 5, also near R12 on the MP-B. Each readily passes three 30-ga. wires to get the address bits off the bus, and the decoded output lead to pin 4 of IC-6; all accomplished with additional carefully soldered connections to bus pins or IC pins on the bottom of the motherboard. Believe it or not, I found it fairly simple to wrap wire to unturned IC pins with a standard hobbyist wrapping tool. Four or five wraps and more are easy.

Note the differences between -B and -B2 connections. On the -B2 motherboard, A12 is defined by IC-6 already, so pin 2 of the 74LS02 should be connected directly to ground instead of A12.

Earle's design used a 74LS20 four-input NAND where I substituted a 74LS30, since I had one. This is an eight-input NAND, so I tied the inputs together in pairs to make the required four inputs. If you use a 74LS20, you may use pins 1,2,4 and 5 as inputs with pin 6 as an output, or pins 9,10,12, and 13 as inputs with pin 8 as the output. Also, be sure to use low-power logic for the quad NOR (-LS02), since it's connected directly to the address bus. This chip is sold as a NOR, but is drawn here as it functions: AND'ing seven or eight inputs that should all be low. A NOR gate is equivalent to an AND gate with active-low inputs.

This article will describe a method by which one may modify the I/O addressing scheme of his SWTPc motherboard, making it switch-selectable for two ranges, thus enabling unqualified use of either 6800 or 6809 CPU boards.

When I first ordered my MP-09 CPU board from Southwest last spring, I anticipated possible compatibility problems with my 6800 system (MP-B motherboard, not -B2; and MP-A processor, not -A2).

The problem was that the 6809 monitor provided with the board, SBUG-E, addresses I/O starting at $E000, compared to $8000 in the previous 6800 monitors I've used, MIKBUG and SWTBUG.

SWTP describes a procedure for re-mapping I/O (and the DMAF-1 controller) under the heading "Memory Map for the MP-09" in the MP-09 board instructions. They explain that their suggested mods render the system incompatible with the original MP-A and -A2 boards using MIKBUG, SWTBUG or DISBUG. In other words, throw out all your 6800 software. Or come up with some 6800 firmware (read EPROM) that addresses I/O at $E000.

An alternative they offer (to maintain compatibility) is to copy the provided firmware onto a 2716 UVPROM, changing two bits at address $FF79 (from $F1 to $F7). This still 1) limits programmable memory capacity to 48K (compared to 56K), 2) limits the system clock to 1 MHz, and 3) allows only the MF-68 minifloppy, not the 8-in. DMAF. Besides, not everyone has 2716 burning/erasing capability. I do, but didn't feel like tying up a $40.00 EPROM for the sake of two bits out of 16,384.

SWTP Motherboard I/O Address Decoding Circuitry

The address decoding for the eight I/O slots on the SWTP motherboard is done by IC-6, a 74S138 one-out-of-eight decoder which decodes three inputs A,B,and C, and enables (active low) one of eight possible outputs, corresponding to possible binary inputs of 000 to 111. On the MP-B board, A,B, and C are tied to address lines A13, A14, and A15, respectively. The selected output (only one has an output trace on the PC board) is pin 11, the 'Y4' output, so called because it goes low when the three inputs equal binary 100, or four. In actuality, further decoding is done by IC-6. On the MP-B, address line A5 is tied to the G2A enable of the chip (active low). This means to select I/O, address line A15 must be high, and A14, A13, and A5 must be low. Only this combination will send Y4 low, which eventally enables the 8 I/O slots. Binarily, this is 100x xxxx xx0x xxxx. The x's are don't-care bits.

On the MP-B2, the same chip is used, but the three inputs are address lines A14, A13, and A12. The only output with a trace connected is pin 15, 'Y0'. Furthermore, address lines A5 and A15 are tied to the G2A and G1 chip enables (active low and high, respectively). This boils down to IC-6 enabling I/O only when address bits 5,12,13, and 14 are low and A15 is high. Binarily this is 1000 xxxx xx0x xxxx. This resulting difference between the MP-B and -B2 means that the B board decodes I/O from $8000 to $801F, but unfortunately, also from $8040 to $805F, and $8080 to $809F, and so on, right up to $9FC0-$9FDF! The MP-B decoding scheme ignores address bits A12 (thus $9000 works just as well as $8000), and A6 through A11, so a whole lot of memory space is wasted, like 8K less 32 bytes! The addressing just isn't "specific" enough.

The -B2 scheme, "looking for" A14, A13, and A12 = 000, won't decode up into the $9000 range, but still ignores bits A6 through A11, so it will also do a "wrap around" decoding of I/O: $8240-$825F will look just like $8000-$801F. Right up to $8FC0-$8FDF, so more or less only 4K is wasted.

How to rectify this, and recover 4 to 8K of memory space for your use, I'll save for another article. Here, for purposes of moving I/O from $8000 to $E000, we're only concerned with address bits A15,14,and 13.

The trick is to use a SPDT switch to select either of two possible outputs from IC-6. The following mod requires an X-Acto knife (for cutting one trace on the motherboard), three lengths of fine wire, (30-ga. wire-wrap would do), and a SPDT switch.

First, the MP-B. After removing the motherboard from the chassis, turn it over and locate pin 11 of IC-6. It's the only connected pin on this side of the IC. Cut the trace right next to pin 11, breaking the connection to IC-5 pins 5 and 6 (another trace branches off to IC-3, pin 4). Next, locate pin 7 of IC-6. This is the 'Y7' output, which goes low when A15-A13 equal binary seven, 111. These are the upper three bits of $E000. Solder a wire to pin 11 and another to pin 7. Connect these to either side of your SPDT switch. I mounted mine on the rear panel of the cabinet, between the 'D' connector holes. The center pole of the switch should now be connected to the other side of the cut trace near IC-6. I simply tied it to pin 5 of IC-5. IC-5 pins 5 and 6 and IC-3 pin now get their 'low' from either IC-6 pins 7 or 11, enabling I/O at either $8000 or $E000. At the flick of a pinkie.

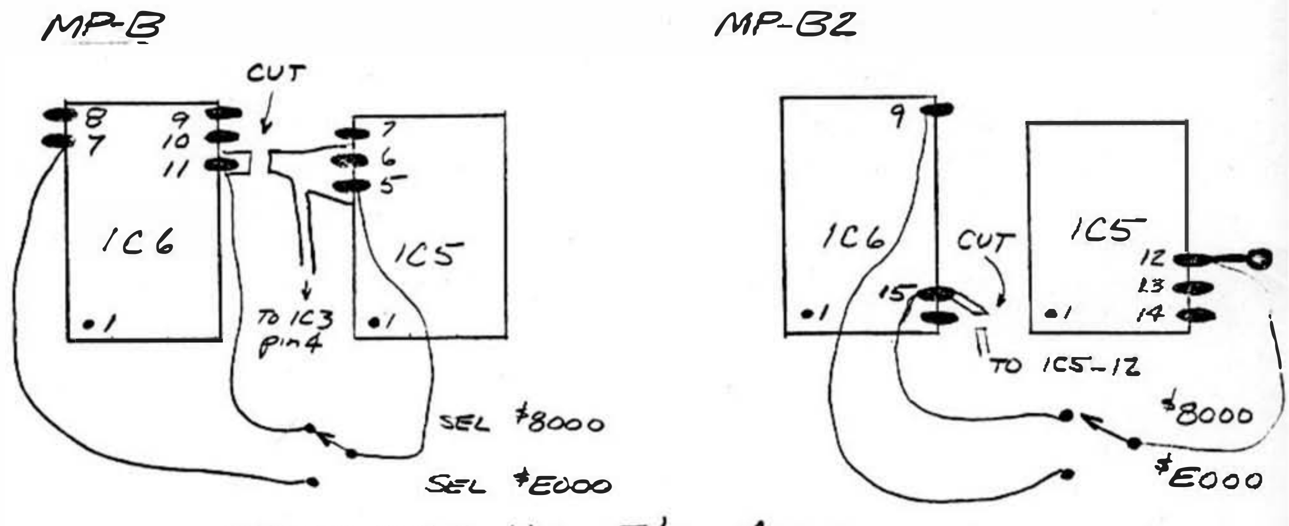
On the MP-B2 motherboard, the two IC-6 outputs will be the existing 'Y0' at pin 15 and the 'Y6' output on pin 9. This latter pin goes low when A14-A12 equal binary 6, 110.
Cut the trace coming from pin 15 of IC-6 right next to the pin. The other side of this trace goes to IC-5, pin 12 and IC-3 pin 4 just like the MP-B. After the trace has been cut, connect the two sides of the switch to IC-6 pins 15 and 9. Connect the center pole of the switch to the other side of the cut trace, either right at IC-5 pin 12 or IC-3 pin 4. IC-5 pin 12 has a feed-thru connected to it, so you could connect there, if you prefer. I routed my three wires under the motherboard to a switch mounted on the rear panel, requiring leads about 15" long, but this is entirely up to the user.

Now, my CPU can be changed from 6800 to 6809 in about 30 seconds, simply removing the 6800 processor board, disconnecting the old Manual Reset leads from the RESET switch, attaching the MP-09's M.RESET leads, connecting the leads to the board, the board to the bus, flick the 'mystery switch' and Go!

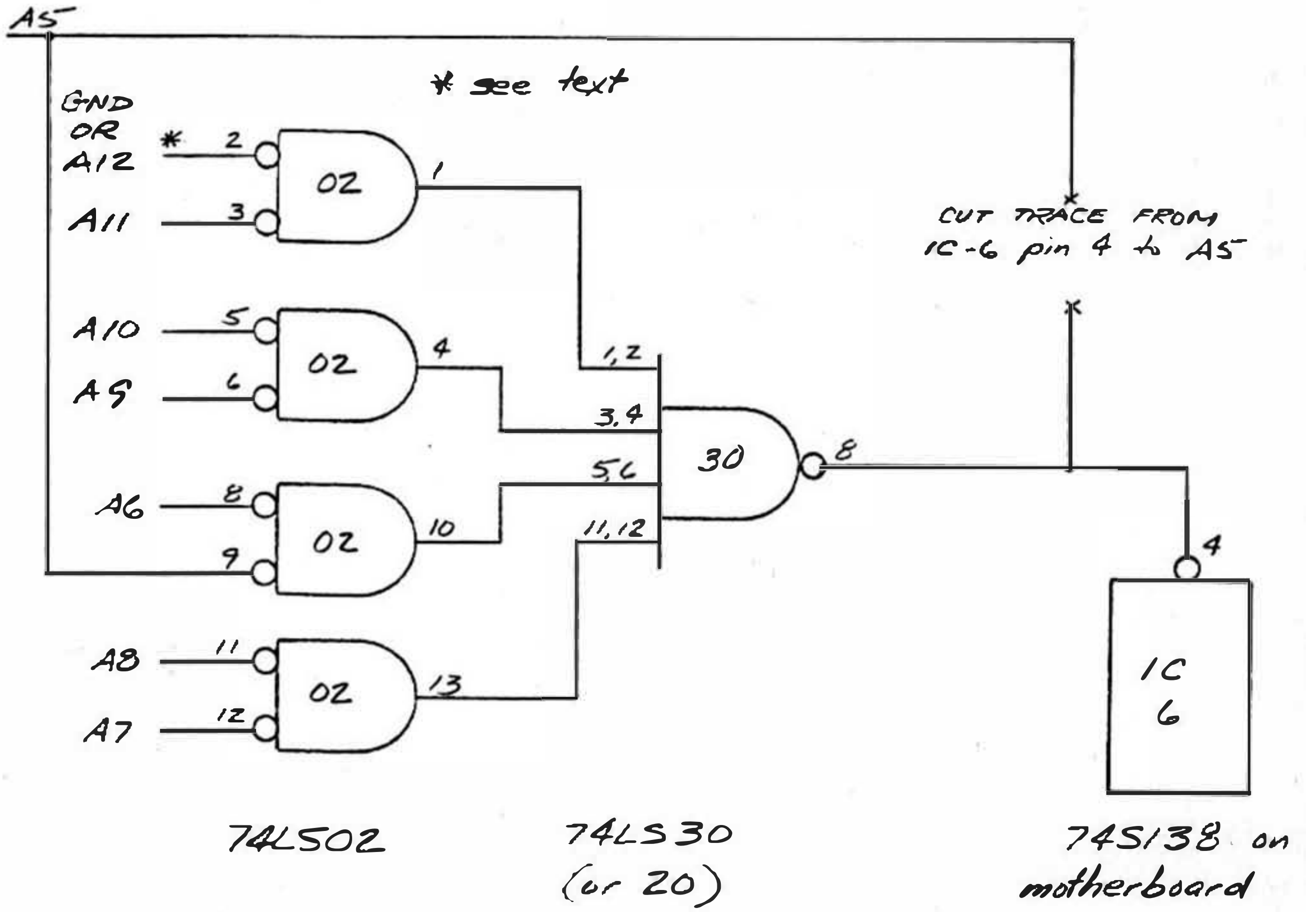
B2 or not B2; 6800 or 6809, there no longer is any question !

RE-ADDRESSING I/O

BOTTOM VIEW OF MOTHERBOARDS



TIGHTENING UP I/O ADDRESSING



FEBRUARY 6, 1980

'68' MICRO JOURNAL
3018 HAMILL ROAD
HIXON, TN 37343

DEAR MR. WILLIAMS,

SINCE I OWN NEITHER A TEXT EDITOR NOR A WORD PROCESSOR, I WROTE THIS LITTLE PROGRAM FOR WRITING, EDITING, AND OUTPUTTING LETTERS OR OTHER TEXT TO MY TELETYPE. IT IS WRITTEN IN COMPUTERWARE SOFTWARE SERVICE'S BASIC V4.3 AND RUNS ON MY SWTPC 6800 COMPUTER.

LINES 1-110 INITIALIZES PROGRAM VARIABLES. LINE=0 DEFEATS LINE PRINT LEADING FUNCTION AND SETTING STRING=64 ALLOWS ONLY 64 CHARACTERS TO BE INPUT TO STRING VARIABLE. LINE 105 MODIFIES BASIC TO ALLOW LEADING SPACES AND COMMAS TO BE USED IN THE INPUT FUNCTION.

LINES 150-170 IS THE INPUT LINES ROUTINE. THERE SHOULD BE NO PROBLEMS WITH THIS ROUTINE.

LINES 200-240 IS THE PRINT ROUTINE. THE PORT= COMMAND IS USED TO ASSIGN THE CONTROL PORT IN CSS BASIC. MY TERMINAL IS ON PORT 1 AND THE TELETYPE ON PORT 3.

THE CONTROL OF THE PROGRAM IS EXECUTED IN LINES 400-591. HERE THE PRINT, EDIT, INPUT OR END FUNCTIONS ARE SELECTED. THERE SHOULD BE NO PROBLEMS HERE AS IT IS FAIRLY STRAIGHT FORWARD.

LINES 600-605 END THE PROGRAM. THE POKE STATEMENTS ARE FOR RETURNING THE USE OF THE COMMA AS A DELIMITER IN STRINGS AND THE ELIMINATION OF LEADING SPACES IN STRINGS. THEREFORE, BASIC WILL WORK NORMALLY FOR ALL OTHER PROGRAMS.

THE CURSOR CONTROL AND LINE # PRINT ROUTINE IN LINES 300-335 MAY CAUSE SOME PEOPLE SOME PROBLEMS. PERHAPS A DESCRIPTION OF WHAT I AM DOING WILL HELP RESOLVE ANY PROBLEMS THAT ARISE. LINE 300 TURNS OFF THE CURSOR; LINE 302 HOMES THE CURSOR TO THE LEFT MARGIN. (I HAVE A SCROLL LOCK ON MY TERMINAL THAT PREVENTS CURSOR MOVEMENT UPWARDS ON HOME OR LINE FEED COMMANDS.) LINE 305 AND 310 BACK SPACE THE CURSOR TO THE EXTREME RIGHT MARGIN AND GETS IT READY TO PRINT THE LINE #. LINES 317-318 SET THE ONE'S DIGIT AND THE TEN'S DIGITS--THEN PRINTS THEM. THE CURSOR IS THEN HOMED AND TURNED ON BY LINES 320-330, AND OF COURSE, LINE 335 RETURNS FROM SUBROUTINE. MY FRIEND'S TERMINAL WILL NOT WORK CORRECTLY WITH THIS ROUTINE UNMODIFIED, SO YOU MAY HAVE TO DO SOME WORK WITH THIS ROUTINE TO GET IT TO WORK ON YOUR SYSTEM.

JERR P. STARZINSKI
POB 9456
YAKIMA, WA 98909

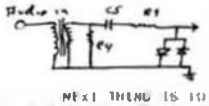
P.S. I WROTE THIS LETTER
USING THE ABOVE
PROGRAM!!!

```
0001 ! LETTER PRINT EDITOR FOR TELETYPE AND SWTPC 6800.
0002 ! CREATED BY JERR P. STARZINSKI---JANUARY 31, 1980.
0100 LINE= 0:STRING=64
0105 POKE( 3654,123):POKE(3794,1):POKE(3795,1):POKE(3796,1)
0110 DIM A$(54):L=1:M=54
0150 FOR X=L TO 54
0152 GOSUB 300::CURSOR POSITION AND LINE # PRINT.
0155 INPUT A$(X)
0160 IF A$(X)="END" GOTO 500
0165 NEXT X
0170 PRINT "END OF PAGE.":GOTO 500
0200 INPUT "PORT=",P:PORT=P
0205 FOR X=L TO M
0210 IF A$(X)="END" THEN PORT=1:GOTO 500
0215 IF P=1 P.A$(X);
0220 IF P=1 GOSUB 300::PRINT LINE #.
0225 IF P=1 P.CHR$(13)::CARRAGE RETURN.
0230 IF P=3 P.A$(X)::PRINT OUT TO TELETYPE.
0235 NEXT X
0240 PORT= 1:GOTO 500
0300 PRINT CHR$(06);
0302 PRINT CHR$(16);
0305 FOR Z=1 TO 3:P.CHR$(08);
0310 NEXT Z
0315 IF X<10 THEN D=X+48:P.CHR$(D);
0316 IF X<10 GOTO 320
0317 LET X$=STR$(X):T$=LEFT$(X$,1):T=VAL(T$)+48:P.CHR$(T);
0318 LET D$=STR$(X):D$=MID$(D$,2,1):D=VAL(D$)+48:P.CHR$(D);
0320 PRINT CHR$(16);
0330 PRINT CHR$(06);
0335 RETURN
0400 INPUT "LINE # TO BE EDITED",X
0405 PRINT CHR$(13):P.CHR$(13):P.A$(X);
0410 GOSUB 300:P.CHR$(13):P.CHR$(13):P.CHR$(13)
0415 PRINT "ENTER CORRECT LINE.":GOSUB 300:INPUT A$(X)
0425 GOTO 500
0500 INPUT "DO YOU WISH TO PRINT(1), EDIT(2), INPUT(3), OR END(4)",B
0505 ON B GOTO 560,400,550,600
0550 INPUT "START INPUTTING WITH LINE #",L:GOTO 150
0560 INPUT "PRINT ALL (1); PRINT FROM L TO M (2); PRINT ONE (3)",R
0565 ON R GOTO 570,580,590
0570 LET L=1:M=54:GOTO 200
0580 INPUT "STARTING LINE",L
0581 INPUT "ENDING LINE",M
0582 GOTO 200
0590 INPUT "LINE TO BE PRINTED",L
0591 LET M=L:GOTO 200
0600 POKE( 3654,44):POKE(3794,189):POKE(3795,9):POKE(3796,99)
0605 END
```

THE FCC HAS IN THE PAST FEW DAYS MADE SOME CHANGES TO PART 97.69 OF THE AMATEUR REGULATIONS TO ALLOW THE USE OF ASCII ON THE AMATEUR BANDS. THESE RULE CHANGES INCLUDE THE BAUD RATE THAT IS ALLOWED ON EACH BAND OF FREQUENCIES. THEY ARE :

3.5 MHZ TO 21.25 MHZ	300 BAUD
28 MHZ TO 225 MHZ	1200 BAUD
420 MHZ AND ABOVE	19.6K BAUD

SO WITH THIS NEWS DIG OUT THE OLD AC-30'S BECAUSE HERE I S
SOME EASY MODES THAT WILL GET THE THING ON THE AIR AND BACK INTO USE WITH OUT SLOWING DOWN YOUR COMPUTER. TAKE OFF THE TOP AND TURN IT SO THE SWITCH'S ARE TO YOU. LOOKING INTO IT ON THE FRONT RIGHT THERE IS A MOLEX EDGE CONNECTOR. WE ARE LOOKING FOR THE 5TH PIN FROM THE RIGHT (IT IS AUDIO INPUT).
THE TRACE THAT GOES TO R8 AND C5 WILL BE CUT AND A AUDIO OUTPUT TRANSFORMER WILL BE INSERTED IN THIS LINE TO GROUND.



THIS TRANSFORMER IS IN PARALLEL WITH R8. IT WILL RAISE THE AUDIO TO A HIGHER LEVEL SO THE AC-30 WILL HAVE MORE TO WORK WITH AND IT IS NEEDED .

NEXT THING IS TO BRING THE SPEED UP TO 1200 BAUD OR WHAT EVER SPEED YOU ARE USING NOW OR PLAN TO USE SOMETIME LATER. STILL LOOKING AT THE BOARD ON THE TOP AND FROM THE FRONT. THERE IS ANOTHER MOLEX EDGE CONNECTOR IN THE LEFT REAR MARKED "COMP". NOW SOLDER A WIRE FROM THE SECOND FROM THE LEFT (TO COMP 16X CLOCK OUTPUT) TO THE MIDDLE EDGE CONNECTOR 4TH FROM THE RIGHT (TO TERMINAL 16X CLOCK INPUT). THIS WILL NOW LET THE TERMINAL CLOCK CHANGE THE SPEED OF THE COMPUTER.

OK NOW REPLACE THE TAPE RECORDER WITH YOUR TRANSMITTER AND RECEIVER AND LOOK FOR ME ON THE AIR. WE CAN SWAP SOME LIES AND SOME PROGRAMS...

K7ZMU AL BAKELER
304 E 1ST PLACE
KENNEWICK WA 99336

al Bakel

1125 Lois Drive
Cincinnati, Ohio 45237
16 February 1980

'68' Micro Journal
3018 Hamill Road
P. O. Box 849
Hixson, Tennessee 37343

Gentlemen:

With regard to my article "Bookkeeping" in February's '68' Micro Journal, a few lines were inadvertently omitted in the paste-up, and I made two errors in the conversion section.

Page 20, column one, insert after the second line:
"However, no system, no matter how accurate, will be used..."

Page 22, column one, the third paragraph from the bottom:

"...back into the matrix (shows it). The disadvantage is that you have to know the account number, so keep your account chart handy."

Page 22, column two, the destination drive is omitted in the second last paragraph.

"COPY,0.EXEC.CMD,1
 COPY,0.COPY.CMD,1".

There are also two coding errors in the BASIC programs, as I discovered at year-end. Insert (or change):

```
CASH.BAS:      1255 IF M = 0 THEN M = 1200
               1343 IF D >1231 THEN D = D - 1200

GPNT.BAS;      1290 IF P > = 0 THEN PRINT "PROFIT FOR YEAR = $";
```

Thank you for printing the article, and keeping us all up to date with other developments in the 68XX world.

William R. Stark

E. BERG PUBLICATIONS

622 East Third
Kimball, NE 69145

The January-December 1979 PERIODICAL GUIDE FOR COMPUTERISTS indexes over 2000 articles from 20 personal computing and professional electronic publications. Articles, editorials, book reviews and letters from readers which have relevance to the Personal Computing field are indexed by subject under 110 categories. A list of the authors is cross-referenced by subject to aid in locating articles. The 80 page book is available for $5.95 from E. Berg Publications, 622 East Third, Kimball, NE 69145 or from local computer stores. A free brochure which describes the book in more detail is available on request. Editions covering 1976, 1977, and 1978 are available for $5.00 each.

SOME BASIC QUICKIES

David Eagle
3330 S Garland Way
Lakewood, CO 80227

```
5000 REM SUBROUTINE CUBIC
5005 REM SOLUTION OF THE CUBIC EQUATION
5010 REM A*X*X*X + B*X*X + C*X + D = O
5015 REM INPUTS, EQUATION COEFFICIENTS, A,B,C,D
5020 REM OUTPUT, ROOTS OF THE CUBIC EQUATION, X1,X2,X3
5025 REM
5030 P= B/A:Q= C/A:R= D/A:E= 1/3
5035 A1= (3*Q-P*P)/3:B1= (2*P*P*P-9*P*Q+27*R)/27
5040 D1= A1*A1*A1/27+B1*B1/4:IF ABS(D1) < 1E-10 THEN D1=O
5045 ON (2+SGN(D1)) GOTO 5050,5070,5080
5050 EO= 2*SQR(-A1/3):C1=-B1/(2*SQR(-A1*A1*A1/27))
5055 S1= SQR(1-C1*C1):GOSUB 9000
5060 Z1= EO*COS(P1/3):Z2= EO*COS(P1/3+2*PI/3)
5065 Z3= EO*COS(P1/3+4*PI/3):GOTO 5095
5070 Z1= SGN(-B1/2)*(2*ABS(-B1/2)^E)
5075 Z2= SGN(B1/2)*(ABS(B1/2)^E):Z3=Z2:GOTO 5095
5080 T1= -B1/2+SQR(D1):T2= -B1/2-SQR(D1)
5085 Z1= SGN(T1)*(ABS(T1)^E)+SGN(T2)*(ABS(T2)^E)
5090 PRINT "X1 REAL, X2,X3 COMPLEX"
5095 X1= Z1-P/3:X2= Z2-P/3:X3= Z3-P/3:RETURN
5100 REM
9000 REM ATAN3 SUBROUTINE
9005 REM INPUTS, S1= SIN(P1), C1=COS(P1)
9010 REM OUTPUT, P1= ATAN3(S1/C1), O=< P1 <=2*PI
9015 IF ABS(S1) < 1E-10 THEN P1= O:RETURN
9020 P1= (2-SGN(S1))*PI/2:IF ABS(C1) < 1E-10 THEN RETURN
9025 P1= P1+SGN(S1)*SGN(C1)*(ABS(ATAN(S1/C1))-PI/2):RETURN
```

```
9000 REM  OLUTION OF SYSTEM OF LINEAR EQUATIONS SUBROUTINE
9005 REM INPUTS
9006 REM MATRIX A, M ROWS BY N COLUMNS
9007 REM VECTOR B, DIMENSION M
9010 REM OUTPUT
9011 REM VECTOR X, DIMENSION N
9012 REM SOLUTION TO SYSTEM, (A)*X= B
9015 ON (2+SGN(M-N)) GOTO 9020,9040,9060
9018 REM
9020 REM UNDER-DETERMINED CASE, M < N
9025 FOR I=1 TO M:FOR J=1 TO M:S= O:FOR K=1 TO N
9030 S= S+A(I,K)*A(J,K):NEXT K:A1(I,J)= S
9035 NEXT J:NEXT I:GOTO 9055
9038 REM
9040 REM EXACTLY DETERMINED CASE, M = N
9045 FOR I=1 TO M:FOR J=1 TO N:A1(I,J)= A(I,J)
9050 NEXT J:NEXT I
9055 L=M:L1=M+1:FOR I=1 TO N:A1(I,L1)=B(I):NEXT I:GOTO 9090
9058 REM
9060 REM OVER-DETERMINED CASE, M > N
9065 L= N:L1= N+1:FOR I=1 TO N:FOR J=1 TO N:S= O
9070 FOR K=1 TO M:S= S+A(K,I)*A(K,J):NEXT K
9078 A1(I,J)= S:NEXT J:NEXT I:FOR I=1 TO N:S=O
9080 FOR K=1 TO M:S= S+A(K,I)*B(K):NEXT K
9085 A1(I,L1)= S:NEXT I
9090 GOSUB 9140
9120 ON (2+SGN(M-N)) GOTO 9125,9135,9135
9125 FOR I=1 TO N:S= O:FOR K=1 TO M:S= S+A(K,I)*A1(K,L1)
9130 NEXT K:X(I)= S:NEXT I:RETURN
9135 FOR I=1 TO N:X(I)= A1(I,L1):NEXT I:RETURN
9138 REM"
9140 REM WORKING MATRIX SUBROUTINE
9145 FOR N1=1 TO L:I1= N1
9150 FOR I=N1 TO L:IF ABS(A1(I,N1)) > ABS(A1(I1,N1)) THEN I1= I
9155 NEXT I:FOR J=N1+1 TO L1:O= A1(I1,J)/A1(I1,N1)
9160 A1(I1,J)= A1(N1,J):A1(N1,J)= O:NEXT J:A1(I1,N1)= A1(N1,N1)
9165 FOR I=1 TO L:IF I=N1 GOTO 9175
9170 FOR K=N1 TO L1:A1(I,K)= A1(I,K)-A1(I,N1)*A1(N1,K):NEXT K
9175 NEXT I:NEXT N1:RETURN
```

Hans-Georg Hupp
Rudolf-Breitscheid-Str.
2970 Emden

BASIC - Statements "!" and "?"

There is a command included in the -BK- BASIC [illegible] that may be very important in some cases. [illegible] is only one control- port (port#1) and one terminal connection. For these systems the command "[illegible]" can be deleted, and substituted by the [illegible] typing time, the statement "!" means the same as PRINT, the statement "?" the same as "INPUT". The change in the BASIC is easily done:

```
Ø326  [illegible] 4E4A
Ø32A  [illegible] 3F ØØ 1A[illegible]
```

The address 1A49 is the address of the PRINT-handling-routine, the address 1A[illegible] the input- handling- routine.

RT-68 Patch To Remote Control For SWTP-BASIC

If you use the RT-68- Monitor in a non SWTP-6800 computer system and without a RS-232 cassette- interface you may have difficulties with SWTP- software such as the SWTP- BASIC Vers. 2.3 . All things are going well except the control of the cassette- recorder for program storage. The ACIA for the cassette- interface is at $ 8008 (PORT#Ø). The control PIA at $ 8004 (PORT#1). The remote- control is connected by relay to CB2 of the PIA (CB2=Ø : motor on; CB2=1 : motor off).
Changing the BASIC-program and adding a short routine you can run cassette- storage with the command "SAVE##". The cassette recorder starts, stops and [illegible] automatically. With "LOAD##" you can load programs in the same way.

```
location    data            change to

ØC13        [illegible] CB  7E 1F83      JMP TAIØFF
ØC16        AØ7F            Ø1 Ø1        NOP

Ø14E        1F83            [illegible]  BEXPI[illegible]

Ø1ØC        7E 1E[illegible]D  7E Ø37B   ACIINT

Routine  TAIØFF
1F83        86 13  TAPØFF   LDAA #$13    Read off
1F85        BD 1E 42        JSR  PØOUT
1F88        86 14           LDAA #$14    Punch off
1F8A        BD 1E42         JSR  PØOUT
1F8D        9E 44           [illegible]
1F8F        CE AØ 7F        LDX  #AØ7F
1F92        7E ØC16         JMP

Change of PØINT, PØ[illegible] and PØOUT
1E2D        39        PØINT     RTS
1E2E        C6 15     CTLSA     LDAB #$15 [illegible]
1E30        E7 ØØ               STAB X,Ø
1E32        C6 34               LDAB #$34  Tape out CB2=Ø
1E34        2Ø Ø6               BRA  CTL3[illegible]
1E36        BD E1 [illegible]  TAIN  JSR  [illegible]
1E39        84 7F               ANDA #$7F
1E3B        39                  RTS
1E3C        F7 8Ø Ø7  CTL3[illegible]  STAB [illegible]
1E3F        [illegible]         [illegible]
1E40        39                  RTS
1E41        Ø1                  NOP
1E42        37        PØOUT     PSHB
1E43        CE 8Ø Ø4            LDX  #PORT#Ø
1E46        C1 11               CMPA #$11  Read on
1E48        27 1Ø               BEQ  CTL1
```

```
1E4A     81 12              CMPA  #$12   Punch on
1E4C     27 16              BEQ   CTL5
1E4E     81 14              CMPA  #$14   Punch off
1E5Ø     27 1Ø              BEQ   CTL5
1E52     81 13              CMPA  #$13   Read off
1E54     27 14              BEQ   CTL5
1E56               CTL6     PULB
1E57     7E E1 A6           JMP   OUT1CH
1E5A     C6 14     CTL1     LDAB  #$14   8+1;+1;RTS.Ø
1E5C     E7 ØØ              STAB  X,Ø    ACIA CTL
1E5E     C6 3C     CTL2     LDAB  #$3C   Tape on CB2=1
1E6Ø     E7 Ø1     CTL3     STAB  X,1    CRB
1E62     2Ø F2              BRA   CTL6
1E64     C6 55     CTL4     LDAB  #$55   8+1;+16;RTS=1
1E66     E7 ØØ              STAB  X,Ø    ACIA CTL
1E68     2Ø F4              BRA   CTL2
1E6A     Ø1 Ø1 Ø1           NOP
1E6D     2Ø BF              BRA   CTL5A
```

MEMORY - TEST

If you have over built yourself a memcryboard in a wire-wrapping-technique or by soldering very thin, insulated cables from any pin to the other, you may have to test it. A test by hand is too hard, but your computer can do it very fast and without errors. It is interesting to know not only the location of defective memory or connections, but also the kind of failure. This program indicates 4 failure- possibilities:

1. It is not possible to write zeros in the testing byte.
   (Report: ØØ DISABLED)
2. Defective (shorted or open) adress-line.
   (Report: OPEN OR SHORTED ADR. LINE)
3. Defective (shorted or open) data-input-line.
   (Report: OPEN OR SHORTED DATA LINE)
4. It is not possible to store ones in the testing byte.
   (Report: FF DISABLED)

The memory-test program indicates the failure and prints adress and failure to the terminal. If all memory locations are allright, the program finishes with "END OF JOB".

First you have to store the beginning adress of the memory range you want to test at $ØØØØ and $ØØØ1. The ending adress you must store at $ØØØ2 and $ØØØ3.

External routines (I use the RT-68-EPROM):

```
EØ7E      PDATA1
EØE3      RT-68 CTL
EØC8      OUT4HS
E141      CRLF
```

For MIKBUG- user change location in the program:

```
ØØ78      BD ØØDE                  JSR      CRLF
```

and add a short routine: (CRLF- MIKBUG)

```
ØØDE      CE E19D      CRLF        LDX      #MCL
ØØE1      7E EØ7E                  JMP      PDATA1 (and return)
```

Following locations are used by the program:

```
$ØØØ4,ØØØ5     XBUF     Buffer for X-register
$ØØØ6,ØØØ7     MEMDAD   Modified ending adress
                        (ENDADR + 1)

ØØ1Ø      DE Ø2        START       LDX      dir.,ENDADR
ØØ12      Ø8                       INX
ØØ13      DF Ø6                    STX      dir.,MEMDAD
ØØ15      DE ØØ                    LDX      dir.,BEGADR
ØØ17      86 ØØ                    LDAA     #Ø
ØØ19      A7 ØØ        ZERO        STAA     X,Ø
ØØ1B      A1 ØØ                    CMPA     X,Ø
ØØ1D      26 2C                    BNE      HALT1
ØØ1F      Ø8           H1          INX
ØØ2Ø      9C Ø6                    CPX      dir.,MEMDAD
ØØ22      26 F5                    BNE      ZERO
ØØ24      DE ØØ                    LDX      dir.,BEGADR
ØØ26      6D ØØ        TEST        TST      X,Ø
ØØ28      26 2A                    BNE      HALT2
ØØ2A      86 Ø1        H2          LDAA     #1
ØØ2C      A7 ØØ        WALBIT      STAA     X,Ø
ØØ2E      A1 ØØ                    CMPA     X,Ø
ØØ3Ø      26 2B                    BNE      HALT3
ØØ32      48                       ASLA
ØØ33      24 F7                    BCC      WALBIT
ØØ35      86 FF        H3          LDAA     #$FF
ØØ37      A7 ØØ                    STAA     X,Ø
ØØ39      A1 ØØ                    CMPA     X,Ø
ØØ3B      26 29                    BNE      HALT4
ØØ3D      Ø8           H4          INX
ØØ3E      9C Ø6                    CPX      dir.,MEMDAD
ØØ4Ø      26 E4                    BNE      TEST
ØØ42      CE ØØ7E                  LDX      #O.K.
ØØ45      BD EØ7E                  JSR      PDATA1
ØØ48      7E EØE3                  JMP      RT-68 CTL
ØØ4B      DF Ø4        HALT1       STX      dir.,XBUF
ØØ4D      CE ØØ8A                  LDX      #STRH1
ØØ5Ø      8D 1D                    BSR      SUBHLT
ØØ52      2Ø CB                    BRA      H1
ØØ54      DF Ø4        HALT2       STX      dir.,XBUF
```

Memory-test page 3

```
ØØ59      8D 14                    BSR      SUBHLT
ØØ5B      2Ø CD                    BRA      H2
ØØ5D      DF Ø4        HALT3       STX      dir.,XBUF
ØØ5F      CE ØØB4                  LDX      #STRH3
ØØ62      8D ØB                    BSR      SUBHLT
ØØ64      2Ø CF                    BRA      H3
ØØ66      DF Ø4        HALT4       STX      dir.,XBUF
ØØ68      CE ØØDØ                  LDX      #STRH4
ØØ6B      8D Ø2                    BSR      SUBHLT
ØØ6D      2Ø CE                    BRA      H4
ØØ6F      BD EØ7E      SUBHLT      JSR      PDATA1
ØØ72      CE ØØØ4                  LDX      #XBUF
ØØ75      BD EØC8                  JSR      OUT4HS
ØØ78      BD E141                  JSR      CRLF
ØØ7B      DE Ø4                    LDX      dir.,XBUF
ØØ7D      39                       RTS

Strings:
O.K. ØØ7E
2Ø 45 4E 44 2Ø 4F 46 2Ø 4A 4F 42 Ø4

STRH1  ØØ8A
2Ø 3Ø 3Ø 2Ø 44 49 53 41 42 4C 45 44 2Ø Ø4

STRH2  ØØ9B
2Ø 4F 5Ø 45 4E 2Ø 4F 52 2Ø 53 48 4F 52 54 45 44 2Ø
41 44 52 2E 2Ø 4C 49 4E 45 2Ø Ø4

STRH3  ØØB4
2Ø 4F 5Ø 45 4E 2Ø 4F 52 2Ø 53 48 4F 52 54 45 44 2Ø
44 41 54 41 2Ø 4C 49 4E 45 2Ø Ø4

STRH4  ØØDØ
2Ø 46 46 2Ø 44 49 53 41 42 4C 45 44 2Ø Ø4
```

Happy testing

yours sincerely

3500 Finfeather #1208
Bryan, Tx 77801
2/11/80

Dear Sir;

After a long wait for parts, I finally got my Micro-Chroma 68 up and running. My thoughts now turn to software.

The 6800 user is at a disadvantage, for the software selection is miniscule. I have a need for a tiny basic interpreter (4K) but have been unable to find one. I need one

I can ROM, but will take anything I can find.

Has your magazine ever printed anything relating to this. If so, I'd like to order a reprint. Otherwise, any help you can give will be greatly appreciated.

Sincerely,

Dana W. Cline

February 5, 1980
946 Evans Road
Nashville, Tennessee 37204

Mr Don Williams Sr
'68' Micro Journal
3018 Hamill Road
Hixson, Tennessee 37204

Dear Sir:

CT-64 Home-up Mod

People who use a stock SWTPC CT-64 in its scrolling mode have one problem: the cursor home function is not defined, as explained on page 13 of the CT-64 CRT terminal assembly instructions. Actually, the home-up cursor function can be implemented in scrolling mode by removing one jumper and adding two more. This mod assumes the CT-64 to be set up for operation in scrolling mode only, and that software or switch selection of page mode or scrolling mode is not implemented.

Remove the jumper from point UP to point S on the main board. No jumper is made from UP to point P. Add a new jumper from point UP to pin 10 of IC 3. To home-up under software control, add a jumper from pin 6 of connector J2 to one of the outputs of IC 43 or IC 44. SWTPC software uses control-P, data link escape, for the home-up function.

Very truly yours,

William R Hamblen

William R. Hamblen

Digital Research Computer
of Texas
P.O. Box 401565
Garland, Texas 75040

Feb. 10, 1980

Gentlemen:

I must write this letter of thanks to a company that did more than just conduct business in a good business manner. I'd had seen there memory board for the 6800 ss50 buss reduced $25.00 in price to $275.00 and I'd decided to send for it. While I was waiting for the memory board to arrive they had another price reduction of $25.00. Well I regretted not waiting a little longer to send for the board kit. Well this is the nice part, I received the memory kit and in with the kit was a $25.00 refund. I don't think that in this day and age a company would have return money and was pleasently surprised. I want to thank you again. I will be getting another 16k board when I add my floppy disc during this year.

I have built your board and to date I've had no problems.

Sincerely,

John Morino

John Morino
518 - 85th Street
Brooklyn, N.Y. 11209

Copy: 68 Micro Journal
File

COMPUTERWARE SOFTWARE SERVICES

MEDICAL OFFICE BUSINESS SYSTEM

A Medical Office Business System (MOBS) is now available for the 6800/6809 computer system. MOBS provides an easy and accurate solution for many tedious and time consuming office tasks. The system will maintain patient account records, prepare billing statements, insurance forms, routine correspondence, and present reports for the management and control of a medical office of one or many doctors. MOBS will improve cash flow, increase office staff productivity, and help to increase the volume of appointments.

The system maintains patient account information on disks. After the account number and data for a patient are entered into the system, appointments, office services, and payments can be posted to that account number. The account status can be recalled and reviewed at any time. The system will prepare billing statements at ay time for either specified accounts or for all accounts with an outstanding balance.

Reports are obtained at the close of the business day, summarizing daily activity and account status. These reports provide not only valuable office statistical data but also a check to insure that account data entry into the system is complete and correct. Other reports list appointments for the next (or specified) day, distribution of credits among doctors, and accounts receivables.

The list of services and fee schedule are entered, updated, and maintained by the user. This means that the system is easily tailored to the specific office requirements and does not require that office procedures be disrupted to fit the system.

The system includes a text editor and text processor which can be used as a word processor independently of the business application software. A useful feature of the system is that system generated lists of patient names and account data can be used by the word processor to issue appointment reminders, patient recall notices, birthday letters, or other routine correspondence with patients. The form letter or notice is constructed by the user to meet his particular requirements.

MOBS is designed to run on a 40K 6800/6809 computer with a minimum of a dual 5" disk system. SSB DOS and Computerware Random BASIC are required.

A manual describing the operation and reports of this system is available for $15.00 from COMPUTERWARE - 1512 Encinitas Blvd. - Box 668 - Encinitas, Calif. 92024 - (714) 436-3512 or 436-0282.

```
9000 REM SUBROUTINE TO PRINT NUMBER WITH DECIMAL PLACES ADJUSTED
9010 REM FOR TSC 6 DIGIT BASIC
9020 REM
9030 REM CALL ROUTINE WITH D=NUMBER OF DIGITS TO RIGHT OF DECIMAL POINT
9040 REM
9050 REM - I - IS THE NUMBER TO BE PRINTED.
9060 REM
9070 REM ===============================================================
9080 REM
9090 I$=RIGHT$("          "+STR$(INT(I*10^D+.5)),8)
9100 PRINT LEFT$(I$,7-D)+"."+RIGHT$(I$,D)
9110 RETURN
```

BILL VODALL
P.O. BOX 336
OILMONT, MONTANA 59466

NEW RELEASE

Lucidata

LUCIDATA has announced the release of Version 2 of their P-6800 Pascal System. This product is designed to run on computer systems based on the Motorola 6800/6809 microprocessors and running the FLEX disk operating systems from Technical Systems Consultants.

New features in this version include the data type REAL (i.e. 9 digit precision floating point values), CASEd RECORDS, own TYPE and sub-range of INTEGER definitions as well as the functions CARD, MOD, ABS, SQR, ROUND and TRUNC. LABELs and the GOTO statement have also been added to facilitate conversion of programs from unstructured languages such as BASIC and FORTRAN. Support of the FLEX Random File facility through the non-standard procedure POSITION (filename, Logical Record) is also provided. Despite all these extensions, P-6800 Pascal will still run on a minimum system sufficient to support FLEX. ROMable versions are also available.

Version 2 of the P-6800 Pascal System is normally supplied on a 5" mini floppy diskette in a specified FLEX format for $150. A User Manual and Installation Instructions are included. Special terms are being offered to owners of the Version 1 System wishing to upgrade to Version 2.

Further details may be obtained from
LUCIDATA
Oosteinde 223
2271 EG Voorburg
The Netherlands
Tel. 070-874489

Release 2 of LUCIDATA's P-6800 PASCAL supports the following:

Simple Data Types:	BYTE[0..255], INTEGER[-32767..32767], REAL[±10^±38 to 9-digit precision], CHAR[nul..rubout], BOOLEAN, enumeration and sub range of integer types.

Data Structures:	ARRAY[up to 7 dimensions] of any type, ALFA[packed array 1..6 of CHAR], SET[64 elements], RECORDs, user defined types (STRINGS), FILE[of any type].
Procedures and functions:	Parameters of any type passed by value. Recursion is supported. Nesting of definitions and calls to 15 levels.
Standard Procedures:	READ, READLN, WRITE, WRITELN, RESET, REWRITE, HALT
Standard Functions:	ORD, CHR, PRED, SUCC, EOLN, EOF, ODD, ABS, MOD, SQR
Non-standard Functions and Procedures:	CARD, POSITION*, UNPACK, USER, PEEK, POKE
Statement Types:	BEGIN..END, IF..THEN..ELSE, CASE..OF..END, WHILE..DO, REPEAT..UNTIL, FOR TO/DOWNTO..DO, GOTO..
Boolean Expressions:	BYTE/INTEGER/REAL/ALFA relationals, CHAR/RECORD/SET/ARRAY equality, OR, AND, NOT, IN set or subrange eg['A'..'Z']
Other Features:	Selective control of listing within program by pragmat $L+or- , easy linking to assembler device drivers, virtual memory mode for small systems, random file support , Hexadecimal constants are recognised.

* not in miniFLEX

26 January, 1980.

'68' Micro Journal,
3018 Hamill Rd.,
PO Box 849,
Rixson, Tennessee, 37343.

Gentlemen:

I am really enjoying the Journal. You are doing a super job. Please find enclosed something you may find suitable for inclusion in it. This is the first time I have ever sent anything to a magazine so I am not sure what the procedure for doing such a thing is. I would appreciate your letting me know if this is not proper.

Since the attatched concerns TSC BASIC, is it normal to send a copy to TSC as well, or not?

I have also modified SWTP BASIC (2.0) for use on the modified D2 kit and I can submit it as well if it would be useable. I notice that most of the articles are for people who are fortunate enough to have printers and disk machines. However, there must be a number of your readers, like myself, who haven't been able to afford such luxuries, and who would like information on how to make better use of the equipment they do have. I would like to see more articles for the likes of us.

Also I have added statements such as SET, RESET, CLS, PRINT@, and POINT to the SWTP BASIC, so that I can run programs such as those written for TRS-80 type graphics, and I would be glad to submit information on this as well.

Yours very truly,

Dick McIlroy,
2107 Gary Cres.,
Burlington, Ontario,
L7R 1T1.

Last October I ordered a BASIC on cassette tape from TSC, and I want to state that I was very impressed by the promptness with which the order was filled. I had the tape and instruction book back within a week of sending in the order. And across the border too!

My computer is a Motorola D2 kit which has been modified with an MEK6800D2 motherboard, an MEK68R2 video interface, an Electrohome monitor, and a Cherry 'PRO' Keyboard. Since the TSC tape is formatted for a machine using the MIKBUG monitor and the D2 kit, using the CRTBUG monitor, expects to see tapes formatted in the JBUG format, I had to write the following loader to get the BASIC into my machine. I have a 16K memory addressed from $0000 to $3FFF, so I have relocated the 256 bytes of on-board memory on the D2 kit to $4000 to $41FF, so this is where I put the loader

```
                    ACIAS   EQU  $8008        ACIA Status
                    CRLF    EQU  $E443        Print 'CR' and 'LF'
                    INCHR   EQU  $E13D        Input a Chr. from Tape
                    DECØDE  EQU  $E494        Print a Chr. on the Screen
                    PDATA1  EQU  $E3D9        Print a Message on the Screen
                    CØNTRL  EQU  $E09B        CRTBUG Control Routine
                    LDCMSG  EQU  $E105        Load Complete Message (my label)
                    ØUT2A   EQU  $E42B        Print 2 Hex Digits
                    NAS     EQU  $A048        No Auto Start (my label)

          4000              ØRG  $4000
4000  86 10         HXLDR   LDAA #%00010000
4002  B7 8008               STAA ACIAS        Ins the ACIA for +1

4005  BD E443       LØAD1   JSR  CRLF         Print 'CR' & 'LF'
400   BD E13D               JSR  INCHR        Get a Chr. from Tape
400B  81 53                 CMPA #'S
400D  26 F9                 BNE  LØAD1        Not an 'S'
400F  BD E494               JSR  DECØDE       Print 'S'

4012  BD E13D               JSR  INCHR        Get another Chr. from Tape
4015  81 39                 CMPA #'9
4017  27 38                 BEQ  LØAD4        Is 'S9' - End of Load
4019  81 31                 CMPA #'1
401B  26 E8                 BNE  LØAD1        Not 'S9' or 'S1'

401D  BD E494               JSR  D CØDE       Print '1'
4020  7F 4057               CLR  CKSUM        Clear Checksum
4023  8D 36                 BSR  BYTE         Get a Byte from Tape
4025  80 02                 SUBA #2           Adjust Byte Count
4027  B7 4058               STAA BYTCNT

402A  8D 2F                 BSR  BYTE
4020  B7 4059               STAA XHI          Hi Byte of Address
402F  8D 2A                 BSR  BYTE
4031  B7 405A               STAA XLØ          Lo Byte of Address
4034  7E 40AC               JMP  LØAD1A       End of TSC tape?

4037  8D 22         LØAD2   BSR  BYTE         Get DATA Byte
4039  7A 4058               DEC  BYTCNT       Reduce Byte Count
403C  27 05                 BEQ  LØAD3        Data Block Done
403E  A7 00                 STAA 0,X          Store Data
4040  08                    INX               Bump Pointer
4041  20 F4                 BRA  LØAD2

4043  7C 4057       LØAD3   INC  CKSUM        Adjust Checksum
4046  27 BD                 BEQ  LØAD1        Next Data Block
4048  CE 407B               LDX  #CKERR       Checksum Error
404B  BD E3D9               JSR  PDATA1       Print Error Message
404E  7E E09B               JMP  CØNTRL       Exit to CRTBUG

4051  BD E494       LØAD4   JSR  DECØDE       Print '9'
4054  7E E105               JMP  LDCMSG       Load Complete

 057       0001     CKSUM   RMB  1            Checksum
4058       0001     BYTCNT  RMB  1            Byte Count
4059       0001     XHI     RMB  1            Address High Byte
  5A       0001     XLØ     RMB  1            Address Low Byte

405B  8D 13         BYTE    BSR  INHEX        Get 1 Hex Chr.
405D  48                    ASLA
405E  48                    ASLA              Raise to Upper Nybble
405F  48                    ASLA
4060  48                    ASLA
4061  16                    TAB               Save in B Acc.
4062  8D 0C                 BSR  INHEX        Another Hex Chr.
4064  1B                    ABA               Combine Chrs.
4065  16                    TAB
4066  FB 4057               ADDB CKSUM        Add to Checksum
4069  F7 4057               STAB CKSUM
406C  BD E42B               JSR  ØUT2A        Print 2 Hex Chrs. to Screen
406F  39                    RTS

4070  BD E13D       INHEX   JSR  INCHR        Get a Chr. from Tape
4073  80 30                 SUBA #$30         De-ASCII-ize
4075  2B 0F                 BMI  HEXERR       Not a Hex Chr.
4077  81 09                 CMPA #9
4079  2F 0A                 BLE  RTS1         0-9 ØK
407B  81 11                 CMPA #$11
407D  2B  7                 BMI  HEXERR       Not a Hex Chr.
407F  81 16                 CMPA #$16
4081  2E 03                 BGT  HEXERR       Not a Hex Chr.
4083  80 07                 SUBA #7           A-F ØK
4085  39            RTS1    RTS

4086  CE 408F       HEXERR  LDX  #HXERR       Hex Chr. Error
4089  BD E3D9               JSR  PDATA1       Print Error Message
408C  7E E09B               JMP  CØNTRL       Exit to CRTBUG

      408F     0A      HXERR   FCB  $A,$D      'CR' & 'LF'
      4090     0D
      4091     48              FCC  "HEX ERRØR"
      4092     45
      4093     58
      4094     20
      4095     45
      4096     52
      4097     52
      4098     4F
      4099     52
      409A     04              FCB  4

      409B     0A      CKERR   FCB  $A,$D      'CR' & 'LF'
      409C     0D
      409D     43              FCC  "CHECKSUM "
      409E     48
      409F     45
      40A0     43
      40A1     4B
      40A2     53
      40A3     55
      40A4     4D
      40A5     20
      40A6     45              FCC  "ERRØR"
      40A7     52
      40A8     52
      40A9     4F
      40AA     52
      40AB     04              FCB  4
```

```
40AC  FE 40 9  LØAD1A LDX  XHI       Address to X-Reg.
40AF  8C A048         CPX  [illegible]
40B2  27 03           BEQ  LØAD1B    Don't allow tape to auto-
                                     start BASIC as $A048-$A049
                                     is used by CRTBUG for other
                                     things.
40B4  7E 4037         JMP  LØAD2     Not end of Tape
40B7  7E E105  LØAD1B JMP  LDCHGO    Load Complete
                      END
```

Once I was able to load the TSC tape, then I had to figure out what I had to change in BASIC so that it would run on my machine. Here are the changes:

TSC BASIC uses Control-X to delete an errored input line and since CRTBUG uses Control-X to turn the cursor on and off, I was continually losing my cursor when I deleted a line. Therefore I changed DELCH ($004A) from $18 to $01 so that I could use Control-A instead to delete a line. Then I changed ($037E) from $58 to $41 so that the delete error message would print 'uparo-A' instead of 'uparo-X'.

I left TSC's end of memory MEMEND ($0040) at $3FFF so that I could use the on-board memory from $4000 to $41FF for patches.

The Input/Output Vectors were set as follows:

```
0106  7E C008  EXIT   JMP  CØNTRL    Exit to CRTBUG
0109  7E A043  INCH   JMP  INCHY     Patch to display incoming Chrs.

010F  7E E494  ØUTCH  JMP  DECØDE    Output to the Screen
0112  7E 4020  TINCH  JMP  TINCHX    Patch for In From Tape
0115  7E 4032  TØUCH  JMP  TØUCHX    Patch for Out To Tape
```

Since the keyboard for this computer comes in through a PIA at $8044, I had to modify checking for a Control-C as follows:

```
01C7  CE 8044  CNTRLC LDX  #PIAIAD
01CA  A6 01           LDAA 1,X       Get PIA Status
01CC  48              ASLA
01CD  24 08           BCC  RTS2      No Data Right Now
01CF  A6 00    CTRLC1 LDAA 0,X       Get PIA Data and check for
                                     Control-C
```

and also:

```
01F8  CE 8044  HUH1   LDX  #PIAIAD
01FB  A6 01           LDAA 1,X       Get PIA Status
01FD  48              ASLA
01FE  24 02           BCC  *+4       No Data Right Now
0200  8D CD           BSR  CTRLC1    Check for Control-C
```

And also:

```
0252  CE 8044         LDX  #PIAIAD
0255  A6 01           LDAA 1,X       Get PIA Status
0257  48              ASLA
0258  24 03           BCC  *+5       No Data Right Now
025A  BD 01CF         JSR  CTRLC1    Check for Control-C
```

Because of the way the initialize routine seeded the Random Number Generator with the same number every time a program was started, the program always began with the same number series. So I made the following change to the initialize routine to modify bits in the seed instead:

```
02D7  96 F2    INZ1   LDAA RANDØM+1
02D9  8A 22           ØRAA #%001 010
02DB  97 F1           STAA RANDØM
02DD  84 DB           ANDA #%11011011
02DF  97 F3           STAA RANDØM+2
```

To get my cursor to literally back up on the screen when I did a Backspace, I put a patch in the Backspace Routine:

```
0509  7E 404F  BCKSPC JMP  PATCH3
```

To get my ACIA for the Tape In and the Tape Øut initialized properly, I had to put the following patches in the SAVE and LØAD routines:

```
      0719  BD 4000  SAVE   JSR  PATCH1
and   078C  BD 4017         JSR  PATCH2
```

Following are the patches referred to above:

```
            4000                 ØRG  $4000
            800A   ACIAS         EQU  $800A    ACIA Status Reg.
            0118   TAPEØN        EQU  $0118    BASIC Turn Tape Øn Vector
            0115   TØUCH         EQU  $0115    BASIC Tape Øut Vector
            01C7   CNTRLC        EQU  $01C7    BASIC Check for Control-C
            E13D   INCHR         EQU  $E13D    CRTBUG Get a Chr. from Tape
            E494   DECØDE        EQU  $E494    CRTBUG Øutput to the Screen
            E1A8   ØUTCO         EQU  $E1A8    CRTBUG Øutput to Tape
            E503   INCH1         EQU  $E508    CRTBUG Get a Chr. from Keyboard
            004C   ECHØ          EQU  $004C    BASIC Echo Inhibit
            A048   CCØUNT        EQU  $A048    CRTBUG Chr. Count on a Line
            A046   CURPØS        EQU  $A046    CRTBUG Cursor Position
            E4F4   UPDATE        EQU  $E4F4    CRTBUG Update Controller Registers
            299F   INPBUF        EQU  $299F    BASIC Input Buffer Start
            04D6   INLIN1        EQU  $04D6    BASIC Input a Line from Keyboard

                   *Initialize ACIA for SAVE.
4000  86 51        PATCH1 LDAA #%01010001
4002  B7 800A             STAA ACIAS     Initialize ACIA for ÷16
4005  BD 0118             JSR  TAPEØN    Turn Tape Øn
4008  BD 400C             JSR  LEADER    Punch Leader
400B  39                  RTS

400C  C6 FA        LEADER LDAB #250      Øutput 250 Leader Chrs.
400E  86 00               LDAA #0         (I do not have auto start and
4010  BD 0115             JSR  TØUCH       stop on my cassette yet, so
4013  5A                  DECB             I have to hear where the
4014  26 F8               BNE  LEADER+2    program starts.)
4016  39                  RTS

                   *Initialize ACIA for LØAD.
4017  86 10        PATCH2 LDAA #%00010000
4019  B7 800A             STAA ACIAS     Initialize ACIA for ÷1
```

```
401C  BD 0118         JSR  TAPEØN    Turn Tape Øn
401F  39              RTS

               *In From Tape.
4020  FF 4030  TINCHX STX  XTEMP     Save X-Reg.
4023  BD 01C7         JSR  CNTRLC    Is there a Control-C?
4026  BD E13D         JSR  INCHR     No - Get Chr. from Tape
4029  BD E494         JSR  DECØDE    Echo to Screen
402C  FE 4030         LDX  XTEMP     Restore X-Reg.
402F  39              RTS

4030     0000  XTEMP  FDB  0

               *Øut to Tape.
4032  FF 4030  TØUCHX STX  XTEMP     Save X-Reg.
4035  36              PSHA           Save A-Acc.
4036  BD 01C7         JSR  CNTRLC    Is there a Control-C?
4039  32              PULA           No - Restore A-Acc.
403A  FE 4030         LDX  XTEMP     Restore X-Reg.
403D  BD E1A8         JSR  ØUTCO     Øutput to Tape
4040  7E E494         JMP  DECØDE    Echo to Screen & Return.

               *To Display Incoming Chrs. on the Screen.
4043  BD E508  INCHY  JSR  INCH1     Get Chr. from Keyboard
4046  7D 004C         TST  ECHØ      Chr. to be Echoed?
4049  26 03           BNE  RTS1X     No - Return
404B  7E E494         JMP  DECØDE    Yes - Øut to Screen & Return
404E  39       RTS1X  RTS

               *Backspace Patch.
404F  FF 4030  PATCH3 STX  XTEMP     Save X-Reg.
4052  7D A048         TST  CCØUNT    Check Chr. Count
4055  27 0D           BEQ  PTCH3A    Already at Beginning of Line
4057  7A A048         DEC  CCØUNT    Reduce Chr. Count
405A  FE A046         LDX  CURPØS
405D  09              DEX            Reduce Cursor Position
405E  FF A046         STX  CURPØS
4061  BD E4F4         JSR  UPDATE    Update Controller Registers
4064  FE 4030  PTCH3A LDX  XTEMP     Restore X-Reg.
4067  8C 299F         CPX  #INPBUF   Beginning of Input Buffer?
406A  27 01           BEQ  *+2       Yes - Get a Chr. for this Line
406C  09              DEX            No - Reduce Buffer Pointer
406D  7E 04D6         JMP  INLIN1    Get another Chr. for this Line
                      END
```

After making these modifications, TSC BASIC worked very well on the D2 kit. It was certainly a lot faster than SWTP BASIC (which I also modified for the D2 kit), especially in the FØR-NEXT loop department.

Now I would like to ask a question! The A8 motherboard has two 86-pin connectors (which now contain the D2 kit and a 16K memory board) and five 60-pin MOXEP type connectors (one of which contains the video driver board). I would like to increase the size of the memory to 32K and I have another 16K memory board which would require another 86-pin socket. I also have a 5-card cage with five 86-pin sockets. Is there a convenient way to connect the two motherboards together, preferably by plugging, or should I replace the two 16K memories with a 32K memory that uses the 60-pin MOXEP connector?

Dick McIlroy,
2107 Gary Cres.,
Burlington, Ontario,
L7R 1T1.

I have long desired a table lookup program for the 6800 processor written using 'good' programming techniques such as being re-entrant and relocatable. I have mentioned my longing to fellow programmers Brad Oestreicher and Bob Meister. The problem was how to pass information as to the table location without using an absolute address. While eating dinner one Tuesday night, Brad looked up from the table and suggested we use a 16 bit offset following the call to pass the information. Bob and I wrote the 'Tuesday Night Dinner Table Lookup' soon afterward. The subroutine is now part of my LEOBUG ROM monitor and is used for ASCII to BCD conversion for my Selectric, address lookup for my keyboard line buffer, etc. To the best of my knowledge the program is re-entrant, since I have run the typewriter under interrupts while using the lookup for another program. I have included a test example to show how to call the lookup. One final note: this program would be only a couple of instructions on a 6809 or PDP-11.

Leo Taylor
18 Ridge Court West
West Haven, Conn. 06516

```
00010                      NAM    LOOKUP


00030                * TUESDAY NIGHT DINNER TABLE LOOKUP
00040                * NOV 23, 1979
00050                *
00060                * OBJECTIVE  A TABLE LOOKUP SUBROUTINE THAT IS
00070                *            RONNABLE, INTERRUPTABLE,RE-ENTRANT
00080                *            AND THE TABLE AND LOOKUP CALL CAN
00090                *            BE RELOCATED ANYWHERE IN MEMORY
00100                *
00110                * ON CALL    A = ITEM NUMBER IN TABLE
00120                *            B = DON'T CARE
00130                *            X = DON'T CARE
00140                *
00150                * ON RETURN  A = DESTROYED
00160                *            B = PRESERVED
00170                *            X = POINTS AT ITEM IN TABLE
00180                *
00190                * PROGRAM EXECUTION RESUMES AFTER OFFSET VALUE
00200                *
00210                * EXAMPLE    JSR LOOKUP
00220                *     LABEL  BRA *+4
00230                *            FDB TABLE-LABEL  OFFSET VALUE
00240                *  <RETURNS HERE, X POINTS AT ITEM IN TABLE>
00250                *
00260                *
00270                * TEST PROGRAM CONVERTS A-J TO 0-9
00280                * EXAMPLE  PRESSING C DISPLAYS C2
00290                *
00300                      OPT    O,NOG
00310 3000                 ORG    $3000

00330 3000 BD E1AC L OP    JSR    $E1AC     GET LETTER INTO A REG
00340 3003 80 41           SUB A  #$41      REDUCE A TO FIRST ENTRY
00350 3005 BD 3800         JSR    LOOKUP    CALL LOOKUP ROUTINE
00360 3008 20 02   HERE    BRA    *+4
00370 300A 03FB            FDB    TABLE-HERE  CALCULATED OFFSET
00380 300C A6 00           LDA A  0,X       FETCH CHAR FROM TABLE
00390 300E BD E1D1         JSR    $E1D1     PRINT RESULTS
00400 3011 20 ED           BRA    L OP      DO IT AGAIN

00420 3400                 ORG    LOOP+$400
00430 3400 30      TABLE   FCC    /0123456789O/



00450 3800                 ORG    LOOP+$800


00470                * TABLE LOOKUP SUBROUTINE
00480                *
00490                * SIZE = 23 BYTES   TIME = 75 CYCLES


00510 3800 37      LOOKUP PSH B              SAVE B
00520 3801 30             TSX
00530 3802 EE 01          LDX     1,X        X POINTS TO OFFSET AFTER CALL
00540 3804 5F             CLR B
00550 3805 AB 03          ADD A   3,X        ADD A REG TO OFFSET
00560 3807 E9 02          ADC B   2,X
00570 3809 30             TSX                X POINTS TO RETURN ON STACK
00580 380A AB 02          ADD A   2,X
00590 380C E9 01          AD  B   1,X        ADD RETURN ADDR  TO OFFSET
00600 380E 36             PSH A
00610 380F 37             PSH B              ADDRESS OF A+TABLE ON STACK
00620 3810 30             TSX
00630 3811 EE 00          LDX     0,X        GET VALUE INTO X
00640 3813 31             INS
00650 3814 31             INS                GET RID OF WORK SPACE
00660 3815 33             PUL B              * STORE B
00670 3816 39             RTS                AND EXIT

00690                     END

TOTAL ERRORS 00000
```

In the January issue of '68' MICRO JOURNAL you published a letter I wrote you about a problem I was having using FLEX. To date I have not received any help from any of your readers. However Mr. Lyle Mays, a computer language teacher at Pittsburg State University here in Pittsburg, KS has given me the answer. Lines 50 and 60 of the inclosed program produce a Record Number and also a Sub Record number from a Logical Entry Number.

```
  10 N=3:  REM  N IS NUMBER OF SUB
RECORDS IN A RECORD (SECTOR)
  20 W=0: REM THESE EXTRA ARE JUST
TO SHOW HOW IT WORKS
  30        PRINT        "RECORD",SUB
RECORD","LOGICAL RECORD"
  40 FOR X=1 TO 10
  50 P=X-INT((X-1)/N)*N-1
  60 Q=INT((X-1)/N)+1
  70 IF W<>Q THEN PRINT:W=Q
  80 PRINT Q,P,X
  90 NEXT X
  RUN
```

```
RECORD  SUB RECORD  LOGICAL ENTRY

  1         0            1
  1         1            2
  1         2            3

  2         0            4
  2         1            5
  2         2            6

  3         0            7
  3         1            8
  3         2            9

  4         0           10
```

This with the added information in the FLEX manual should allow me to do what I want.
Many Thanks!

FRANK C. BARNEY
316-231-1970
425 North Bdwy.
Pittsburg, KS
66762

TRIM, a program for making old BASIC programs compatible with later versions of the SWTP 8K BASIC

Programs saved to cassette under SWTP 8K BASIC Version 1 will not work when reloaded and run under Version 2.2. The problem lies in statements of the form Variable = Expression.

The obvious solution is to reload and run such programs under Version 1, but I wanted to take advantage of the increased speed of Version 2. Another solution is to manually re-enter the offending lines, but I had a fairly large collection of long programs, and wanted to avoid the work and the risk of re-entry.

Fortunately, the differences between the way the programs are stored under the two versions is systematic, permitting a short machine language program, TRIM, to "update" the older programs.

Each statement in SWTP BASIC is stored in the form: (1 delimiter byte,always Ø),(line number, 2 BCD bytes),(2 bytes defining the statement type),(1 byte for the statement length), and then the statement itself. The problem statements have two extra bytes between the line number byte and the statement proper. These bytes, when they occur, always contain 7F and 2Ø. TRIM searches for any occurrence of 7F, subtracts 2 from the line length byte (to account for the 2 bytes to be removed) and then moves the whole of the remainder of the program up two bytes to obliterate the 7F and 2Ø. This process is repeated for all occurrences of 7F.

TRIM could be loaded above the BASIC program, but running the Basic program, or attempting to run it, will wipe out TRIM. The best place to store it is between the interpreter and the BASIC program. This will permit it to remain in place while several BASIC programs are loaded, "cleaned-up" and run.

To create space for TRIM , increase the contents of Ø14E and Ø14F before loading the BASIC program. This forces the interpreter to start allocating memory at the new address stored at Ø14E and Ø14F. Since TRIM is about 46 bytes long, this new starting address should be set at, say, 1F2Ø. Since the BASIC interpreter reserves 256 bytes at the end of memory for a string buffer and machine stack, TRIM should be set to stop short of this area: at 3EFE for 16K of memory, for instance, or as listed in Table 1. Note that this stopping address appears twice in TRIM: enter the appropriate address in both places according to the size of your memory.

To use TRIM, load Version 2 of the interpreter, change the contents of Ø14E and Ø14F to 1F2Ø using the M command of MIKBUG or SWTBUG, then load TRIM starting at 1EE7, again using the M command. Now press G to reach the BASIC command level, and load the old program from cassette. Return to the system level by using the PATCH command and change the contents of AØ48, AØ49 to the starting address of TRIM, i.e. 1EE7, and press G to execute TRIM. After a few seconds, depending on the size of your memory and the number of lines which need to be trimmed, the system will return to the MIKBUG or SWTBUG monitor level. When this happens, use the M command again to change the contents of AØ48, AØ49 to Ø1Ø3, and press G. This will return you to the BASIC command level. The old program should now run, and may be saved to cassette in its new form. If another old program needs trimming, load it from cassette and use the PATCH command to return to the system monitor level. Change the contents of AØ48,AØ49 to 1EE7 and press G to execute TRIM. When the monitor prompt reappears, reload Ø1Ø3 into AØ48,AØ49 and press G. A series of BASIC programs may thus be trimmed and resaved without having to reload TRIM.

Table 1

Memory size:	Stop TRIM at:
12K	2EFE
16K	3EFE
20K	4EFE
24K	5EFE
28K	6EFE
32K	7EFE

Program listing: TRIM

```
1EE5  00 00          TMP1   RMB 2
1EE7  CE 1F20        START  LDX $1F20
1EEA  08             FIND   INX
1EEB  8C 3EFE               CPX $3EFE     FOR 16K MEMORY ONLY
1EEE  27 20                 BEQ  FINI     TO MIKBUG JUMP
1EF0  A6 00                 LDA A 0,X
1EF2  81 7F                 CMP A $7F     BAD BYTE?
1EF4  26 F4                 BNE  FIND
1EF6  09             DECCTR DEX           BACK UP
1EF7  A6 00                 LDA A 0,X     GET COUNT BYTE
1EF9  4A                    DEC A         FIX COUNT
1EFA  4A                    DEC A
1EFB  A7 00                 STA A 0,X     RESTORE COUNT
1EFD  08                    INX           FORWARD AGAIN
1EFE  FF 1EE5        MOVE   STX  TMP1     SAVE ADDRESS
1F01  A6 02          LOOP   LDA A  2,X    MOVE BYTE DOWN TWO
1F03  A7 00                 STA A  0,X
1F05  08                    INX
1F06  8C 3EFE               CPX $3EFE     FOR 16K MEMORY ONLY
1F09  26 F6                 BNE LOOP
1F0B  FE 1EE5               LDX TMP1      GET ADDRESS
1F0E  20 DA                 BRA FIND
1F10  7E E0E3        FINI   JMP $E0E3     TO MIKBUG CONTROL
```

As a result of extensive testing, A,B,C,D, and E were ranked on the Hochstrasser Scale of Overt Intellectual Snobbery, without ties.

In subsequent discussion, A stated, "I know my rank, and that of B - I was higher than he was - but I know none of the others' ranks. If I knew that C was three places higher than D, which is roughly what I would expect, then I would know the rank order of all five of us."

E, who had been listening to A's remarks, has been told no one's place but his own. However, he is confident that D could not be higher than B, and, as it happens, he is right in this surmise. After a pause for reflection, E says, " I can now write down the complete rank order" He does so, and is quite right.

What was their order?

R.C. Mosley
14 Standish Circle
Andover, MA 01810

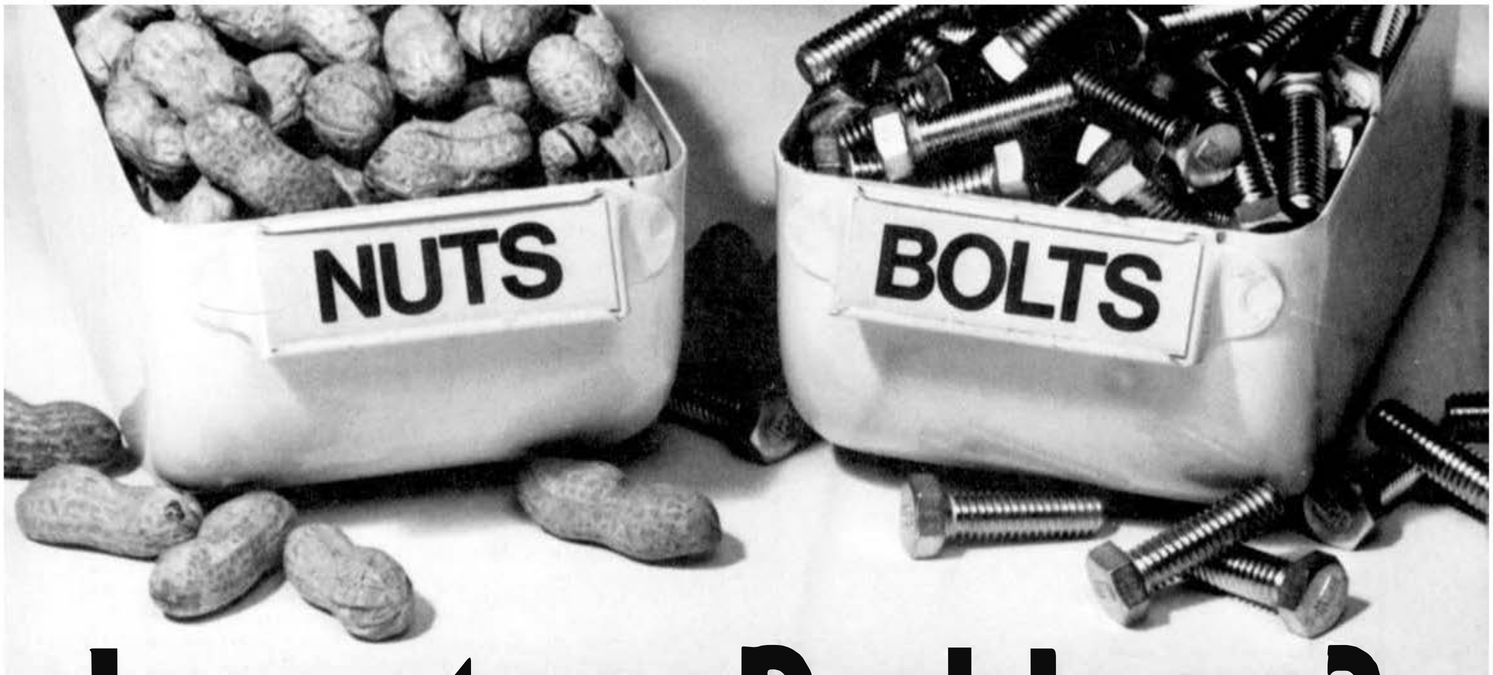
This is the 'BIT-BUCKET'.

After a year plus we still receive material that is of great interest to many and yet not quite large enough to be handled as a full fledged article (?). Most of it (as is a lot of the rest) is photo-reduced to fit on a page with other such material. I have used various headers and yet it is a pain to try to figure what belongs where. So as you have probably figured, I am going to cop-out and put a mixture of goodies about here. This will be called the 'Bit Bucket'.

Here will be found letters, new products, gripes, hints and kinks (both hard and soft), my mutterings and ramblings occasionally and just about anything else I cannot fit elsewhere. As I receive or hear choice bits of gossip or rumors about 68XX products those will fall in the bucket also.

From your input in the past these are well received and so I will just collect them all together and try to drop them in the bucket each month. Let me know what you think.

DMW



Inventory Problems?

Are you having trouble keeping the right nuts and bolts in stock? Since even a simple mistake can cost you time and money, a good inventory system should do more than just count parts. It should tell you exactly what you need, when you need it, where to get it, and how much it will cost.

The MSI Inventory System Seven enables you to maintain a versatile data base for controlling inventory. It lists part number, description, quantity on hand, vendor, cost, selling price, optional pricing, usage levels for previous month, present month, and year-to-date, and much more.

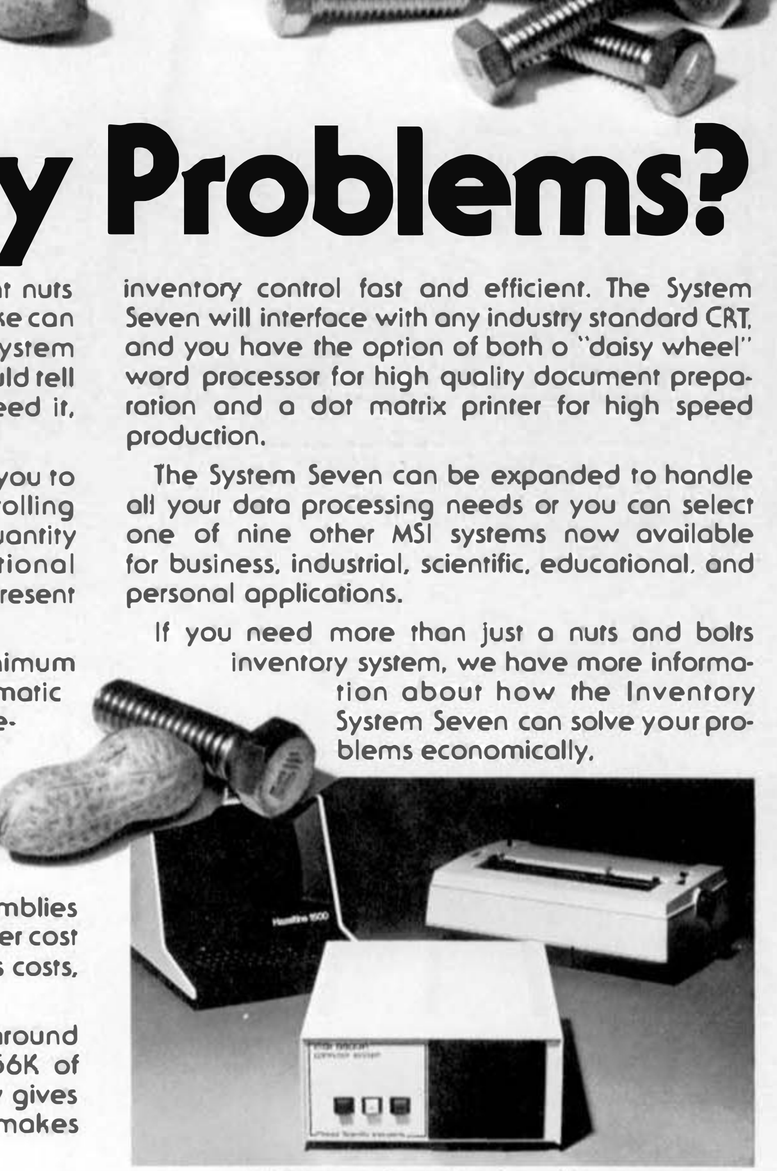
When quantity on hand items reach minimum levels, the System Seven compiles an automatic reorder list. This list can be generated by specific vendor as well as a complete listing of all materials to be ordered.

In addition to the item listing, the Inventory System Seven "bill of materials" provides you with a complete inventory of items used in the manufacture of subassemblies and complete products. It also contains other cost items such as labor costs, total raw materials costs, and miscellaneous costs.

The MSI Inventory System Seven is built around the versatile MSI 6800A Computer with 56K of RAM. An integral dual mini-floppy memory gives you an additional 630K of memory and makes inventory control fast and efficient. The System Seven will interface with any industry standard CRT, and you have the option of both a "daisy wheel" word processor for high quality document preparation and a dot matrix printer for high speed production.

The System Seven can be expanded to handle all your data processing needs or you can select one of nine other MSI systems now available for business, industrial, scientific, educational, and personal applications.

If you need more than just a nuts and bolts inventory system, we have more information about how the Inventory System Seven can solve your problems economically.



MSI Inventory System Seven

MSI

Midwest Scientific

220 W. Cedar, Olathe, Kansas 66061, (913) 764-3273
TWX 910 749 6403 (MSI OLAT), TELEX 42525 (MSI A OLAT)

P-6800 PASCAL

Lucidata

RELEASE 2

P-6800 PASCAL now has REALS ($\pm 10^{\pm 38}$ 9-digit precision)
P-6800 PASCAL now has RECOROS (owntype definitions)
P-6800 PASCAL now supports RANDOM FILES*
P-6800 PASCAL now has a COMPLETELY REVISED MANUAL
P-6800 PASCAL includes lots of SAMPLE DEMO PROGRAMS
P-6800 PASCAL has a SIMPLE USER interface
P-6800 PASCAL is easily linked to your own peripherals
P-6800 PASCAL does NOT require you to buy more hardware
P-6800 PASCAL will still run in ONLY 16K
P-6800 PASCAL requires ONLY ONE SINGLE DENSITY 5" DRIVE
P-6800 PASCAL requires NO SPECIAL TERMINAL
P-6800 PASCAL does NOT force you to abandon FLEX™
P-6800 PASCAL STILL costs only $150 (VISA/Mastercharge)

```
P-6800 PASCAL   is supplied in versions 2.1      miniFLEX
                                         2.2        FLEX 2
                                         2.9 FLEX 9 (6809)
```

FURTHER DETAILS FROM:

LUCIDATA
Oostelade 223
2271 EG VOORBURG
NETHERLANDS

™ FLEX is the trademark of TSC Inc.
* Not In Version 2.1 (miniFLEX)

6800 Software

Hardware, Firmware

This month's Special...

CHECK 'N TAX

Put your 6800 to work balancing your checkbook and preparing your income tax. Check 'n Tax is a set of eleven Basic programs which balance your checkbook each month, cross-check your check stubs against every single bank entry, find and help you correct errors. The result is a year-to-date check file on disk with each entry coded by income tax category. To this you add other income and expense data during the year. At income tax time, you get a printout of all expense and income items, broken down by tax category and ready for use. Available on miniFlex or Flex 2.0 (TM of TSC) disk or on Percom LFD-400 Disk for $40 list -- this month's special price is $35, which includes a free copy of FLOGEN. Specify which Basic and DOS you use.

STAR - KITS

P.O. Box 209, Mt. Kisco, N.Y. 10549

6809—DATA FILE MAINTENANCE

STOP writing dinky little programs for all the one-time changes to one item on a data file. START bringing up new systems without long weeks of programming.

The General Data File Maintenance Program can add, delete, insert, and modify data on any file you have*! The powerful security allows you to restrict modification of data already entered.

This software tool will save you days of programming effort with commands that can list, print or show your data. Some of the many things you might use it for are: inventory files, Customer files, Real Estate Listings, plus many more. Let your imagination run WILD!

You can format the items in many ways with this 6809 Extended BASIC program. Some of the options available are right or left justify, item length, etc.

Order your diskette today for only $49.95! Use Master Charge, VISA or check. Specify diskette size.

Tennessee residents add 6¼% sales tax. Customers outside Canada or USA add $5.00 for air postage and handling. If you wish to order by phone, give us a call at (615) 396-9111.

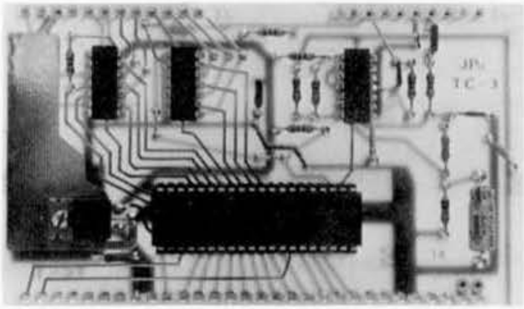
Coming soon VER. 2.1 for TSC Multiuser Basic™

*Record sizes up to 252 bytes.

dp systems

po box 567 collegedale tn 37315

JPC PRODUCTS FOR 6800 COMPUTERS



High Performance Cassette Interface

- **FAST** - 4800 Baud Loads 4K in 8 Seconds!
- **RELIABLE** - Error Rate Less Than 1 in 10^8 Bytes.
- **CONVENIENT** - Plugs Directly Into The **SWTPC.**
- **PLUS** - A Fully Buffered 8 Bit Output Port Provided
- **LOW COST** - $59.95 For Complete Kit.

- **OPTIONAL** - CFM/3 File Manager.

Manual & Listing	$19.95
(For Cassette Add)	$ 6.95

TERMS: CASH, MC or VISA. Shipping & Handling $3.00

JPC products

Order Phone (505) 294-4623
P.O. Box 5615
Albuquerque, N.M. 87185

AAA CHICAGO COMPUTER CENTER

120 Chestnut Lane
Wheeling, IL 60090
(312) 459-0450

6800/09 Specialists
Dealer for Gimix, TSC
Smoke Signal, SWTPC

Microtime 6800 Bare Board & Documentation	$ 35.00
Assembled with parts (push button set)	$ 95.00
Push Buttons & Software set	$105.00
11½ digit Math Package with Fortran Type Formatting	$100.00
Business Basic Version R3 with automatic line renumbering print using, and more	50.00
Fast Basic (by Mark Data Products)	60 .00
Payroll (Process any number of employees: fast)	400.00
Editor-Text Processor Version 2.0 with Mail Labels & Mailing List Specify Smoke, Flex 1.0 or 2.0, MS1, GMXBUG	100.00

(Note: Above Basics have random disc files and were designed for Smoke disk: available on cassette for en additional $10.00)

SEE GIMIX AD ON P. 3

MULTI-USER ON A 56K 6809

Support four totally independent terminals with one computer. Each terminal has access to the services you need most in your classroom or business.

COMMON PILOT - the best CAI system in the world; useful in any application where computer-user dialogue is desirable.

STUDENT BASIC - real numbers, functions, arrays, strings, loops, subroutines, and random disk-file access.

SCREEN-ORIENTED EDITOR - edit lines directly by overtyping, inserting, and deleting; useful for programming and word processing.

Hardware requirements: SwTPC /09 (the cheaper 6809), 56 K, MP-N calculator board, dual floppy disks (either size), one to four MP-S serial interfaces, one to four SwTPC CT-82 terminals.

$250.00 single copy (specify 5" or 8")
(quantity discounts available)

MICROPI
2445 N. Nugent
Lummi Island, Washington 98262

6800/6809 PASCAL

DYNASOFT PASCAL is a cassette based PASCAL subset designed to run on most 6800/6809 systems with 12K or more of memory.

DYNASOFT PASCAL includes most of the control structures of standard PASCAL including IF-THEN-ELSE, CASE-OF-OTHERWISE, WHILE-DO, REPEAT-UNTIL, FOR-TO/DOWNTO-DO, and recursive PROCEDURE's and FUNCTION's. It supports the data types INTEGER, CHAR, BOOLEAN, scalar (user-defined), subrange, pointer and ARRAY. It is built around a one pass compiler which produces fast, compact p-code and comes complete with a line-oriented text editor, p-code interpreter, and program SAVE and LOAD routines. The whole system resides in less than 8K.

The cassette version with manual is priced at $35 plus $3 for postage and handling. Please specify 6800 or 6809.

Dyna
soft
systems

P. O. BOX 51
WINDSOR JCT., N. S.
CANADA B0N 2V0

6800/6809 SOFTWARE

ACCOUNTS PAYABLE #1300
Produce financial reports, print chec s, special control letter. Reports by vendor number, invoice number, aged and history file. Auto sorting of vendor and invoice files. Plus check and pre check journals. **$800.00**

ACCOUNTS RECEIVABLE #1500
Produces financial reports, prints statements, produces reports by customer account number, invoice by customer account number and invoice by invoice number. Print aged report and trial balance. Keeps history file and auto sorting of files. **$600.00**

GENERAL LEDGER #100
Program updates to ledger files and also generates reports on payroll, sales, accounts payable, cash and expense statistics. Balance sheet and profit & loss reports. Information can be generated for year end taxes, 941 and W2 forms. **$595.00**

INVENTORY I, #600
Inventory for a s all company. Produces activity reports for day, mo th and year. Minimum quantity search. Inventory list by class, by vendor or complete with totals and financial report. **$100.00**

INVENTORY II, #700
Produce inventory reports by description or vendor, print activity reports for one day, one month or one year. Quick search by p rt number, produce total inventory and financial report. (For one store) **$200.00**

INVENTORY III, #800
Same as inventory II, #700, but produces reports for eight stor . **$300.00**

SHIPPING/RECEIVING
Produce your business financial report, produce reports on sales, accounts receivable and last purchase by customer. Print mailing labels and print customers bills. **$ 75.00**

MAILING LABLES #100
Print mailing labels from your complete file, for a particular city or state. Use one-part mailing labels. **$ 50.00**

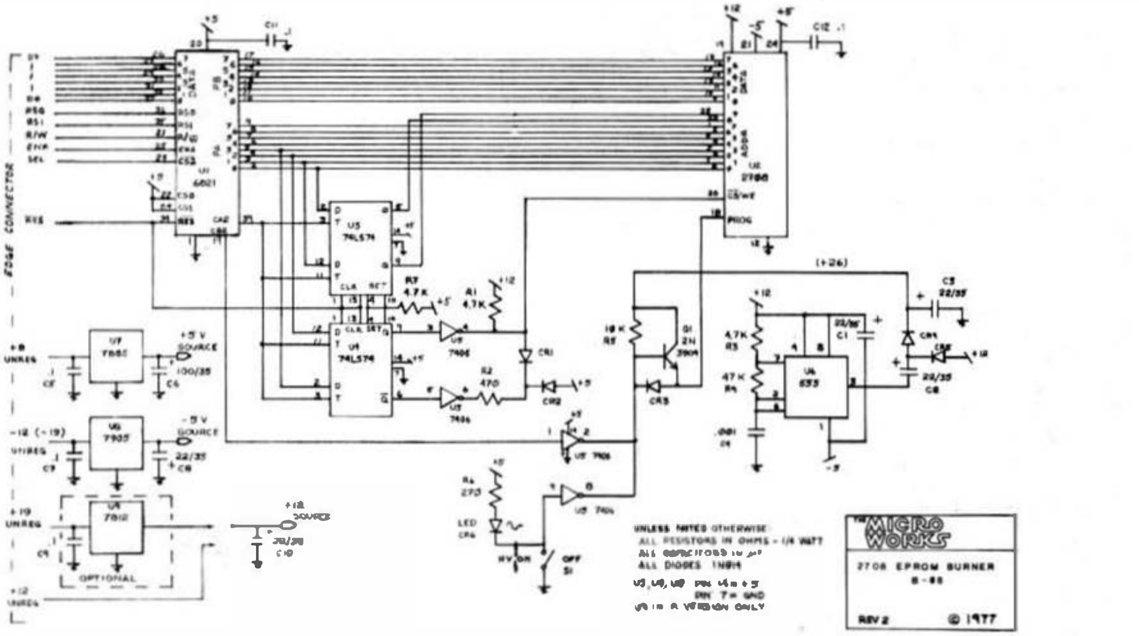
MAILING LABELS #400
Same as #100, but also prints labels by names. Use multiple-part labels. **$125.00**

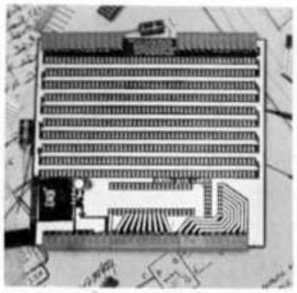
BASIC—0935
For those with applications in SWTP 3.5 BASIC. Runs on 6809 and 6809S. 30% faster and can be used with existing 6800 BASIC Programs. No manual, commands and statements same as SWTPC BASIC 3.5. Ideal to keep you going while changing to new BASIC's. 5¼ or 8 inch 09 Disk, with Renumber routine. **$ 59.95**

****Customized programs for your business requirements****
Charge your order to your Visa or Master Charge

Available from Computer Stores or order direct from:
Omnitronics Inc, 1897 Rt. 33, Concord Square,
Hamilton SQ., NJ 08690

Build Your Own 2708 Programmer!! HERE'S HOW:





And here's the wire wrap board to build it on. The Universal I/O Board has space for a 40-pin wire wrap socket for any of Motorola's 40 or 24 pin interface chips; data and control lines are connected to appropriate edge connector pins. All other bus connections are brought out to a 16 pin socket pad. UIO: $24.95

Software? A stripped down version of our U2708 Utility program was published in *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia*, February Issue, #32. Or the *complete* version of U2708 is available in the B-08 Owner's Manual for only $10.00.

BUT...

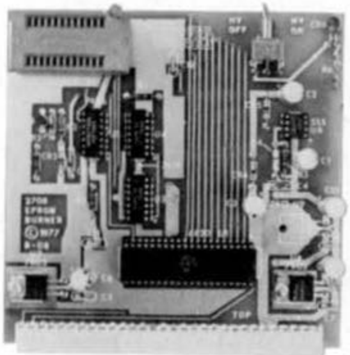
If you're not a dyed in the wool wirewrapper and would like to purchase

The Real Thing

Assembled, Tested, Burned in and

READY TO GO

GOTCHA COVERED



B-08 Eprom Programmer $99.95

U2708 Software in Eprom $29.95

!

THE MICRO WORKS P.O. BOX 1110, DEL MAR, CA 92014 714-942-2400

CURSOR/SCREEN EDITING IS HERE FOR THE SS-50!

Research conducted by a reliable source (MY WIFE) indicates that you spend more time looking at your CURSOR than your own FAMILY! Don't you think it's about time you used the video board and the cursor which cost you good money for something other than a poor excuse for not looking at your family?!

The unique SCREDITOR CURSOR—BASED SCREEN EDITING SYSTEM is just what you need to make your hardware work! Take a look at a few of the features of the SCREOITOR and see what you're missing —

- EVERYTHING HAPPENS ON THE SCREEN — YOU SEE WHAT IS HAPPENING AS IT HAPPENS!
- INSERT CHARACTERS, WORDS, LINES OR EVEN OTHER FILES, AND WATCH IT AS IT HAPPENS — RIGHT ON THE SCREEN!
- FILE SIZE IS ONLY LIMITED BY DISC SPACE!
- PRINT COMMAND PROVIDES HARDCOPY WITH ADJUSTABLE MARGIN TO MATCH YOUR PRINTER!
- AUDIBLE FEEDBACK THROUGH A USER—SUPPLIED NEWTECH BOARD SOUNDS KEYSTROKES AND EDIT—MESSAGES AS THEY OCCUR!
- SELECTABLE TEXT MODE ALLOWS CONTINUOUS TYPING WITHOUT CONCERN FOR MARGINS!
- FULLY UTILIZES THE FLEXIBILITY AVAILABLE ONLY IN A MEMORY—MAPPED VIDEO DISPLAY!
- MOVE LINES AND PARAGRAPHS FROM PLACE TO PLACE IN THE FILE YOU ARE EDITING — EVEN BEYOND THE BLOCK IN MEMORY!
- SET, DISPLAY, CLEAR AND USE TABS LIKE A TYPEWRITER!
- FINO AND CHANGE COMMANDS WAIT TO BE VERIFIED TO HELP STAMP OUT ERRORS!
- IMPORTANT INFORMATION SUCH AS LINE, COLUMN, MODE, ETC. . . , IS CONTINUOUSLY DISPLAYED AND UPDATED ON THE SCREEN!
- EDITING OPERATIONS ARE CURSOR—ORIENTED. THE CURSOR POINTS TO WHERE THE ACTION IS!

The features go on and on! With FOURTEEN major commands, TWO edit modes, TWO major file handling modes and TWENTY—TWO screen operators, the SCREOITOR offers a level of control and convenience never before available to users of the SS—50 buss.

Order your copy of the SCREDITOR for 16 x 64 displays (such as the Thomas Instrumentation Board) OR for 24 x 80 displays (such as SSB VDB-1) to run under SSB DOS68.51X today! To place your order, call or write —

BE SURE to specify the version desired when ordering.

FLEX

FLEX™
Specify Miniflex, Flex 2.0
5" or 8" Disk

ALFORD AND ASSOCIATES
P. O. BOX 6743
RICHMOND, VA. 23230
804-329-3906

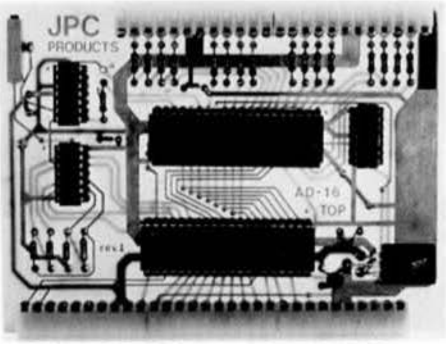
SSB

$99.95

Va. residents add 4%.

VISA and M. C. accepted graciously. Sorry, no COD's. Shipment is normally within three days of receipt of order unless by check. 6809 version available soon for SSB's 6809 DOS.

JPC PRODUCTS FOR 6800 COMPUTERS



USES ONE I/O SLOT

16 CHANNEL A/D BOARD

- 8 BIT DATA
- SOFTWARE CONTROLLED GAIN
- 3300 SAMPLES PER SECOND
- ± 0.7% ACCURACY

COMPLETE KIT: AD-16 $69.95

Terms: Cash, MC or Visa; Shipping & Handling $3.00

JPC products

Order Phone (505) 294-4623
P.O. Box 5615
Albuquerque, N.M. 87185

Model EP-2A-88 EPROM Programmer

Fast as Jackrabbits . . . Well, almost!

In Australia, two rabbits can reproduce over 13 million offspring in 3 years . . . at 105 seconds per 2716, the EP-2A-88 can reproduce 1,892,160 EPROMS in 3 years. Single push button control, the EP-2A-88 checks if EPROMS are erased, programs and verifies. It also checks for defective EPROMS.

Two basic m dels are available. The EP-2A-88-1 will accept Copy (CM) modules for the 2758, and 2716 EPROMS. The EP-2A-88-2 will accept c py m dules for the 2716, 2732 and TMS 2532 EPROMS. Power requirements are 115 VAC 50/60 Hertz at 15 watts.

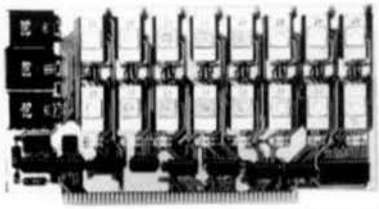
Part No.	Description	Price
EP 2A-88-1	EPROM Programmer	$450.00
EP 2A-88-2	EPROM Programmer	450.00
CM 50	Copy Module for 2716, TMS 2516 EPROMS	25.00
CM 70	Copy Module for 2758 EPROMS	25.00
CM 20	Copy Module for 2732 EPROMS	25.00
CM 40	Copy Module for TMS 2532 EPROMS	25.00
	Non Standard Voltage Option (220 v, 240 v, 100 v)	15.00

Optimal Technology, Inc.
Blue Wood 127, Earlysville, Virginia 22936
Phone (804) 973-5482

DIGITAL RESEARCH COMPUTERS

(214) 494-1505

16K EPROM CARD-S 100 BUSS



$59.95 KIT

FIRST TIME OFFERED!
BLANK PC BOARD - $28

USES 2708's!

Thousands of personal and business systems around the world use this board with complete satisfaction. Puts 16K of software on line at **ALL TIMES!** Kit features a top quality soldermasked and silk-screened PC board and first run parts and sockets. Any number of EPROM locations may be disabled to avoid any memory conflicts. Fully buffered and has WAIT STATE capabilities.

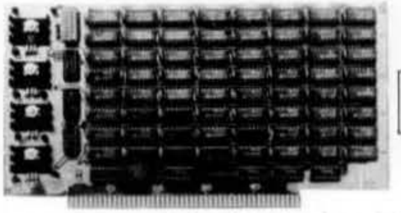
OUR 450 NS 2708'S ARE $8.95 EA. WITH PURCHASE OF KIT

ASSEMBLED AND FULLY TESTED ADD $30

8K LOW POWER RAM KIT-S 100 BUSS

SALE

PRICE CUT!



$119.50 KIT

21L02
(450 NS RAMS!)

Thousands of computer systems rely on this rugged, work horse, RAM board. Designed for error-free, NO HASSLE, systems use.

ASSEMBLED AND FULLY BURNED IN ADD $35

ALL ASSEMBLED BOARDS ARE TESTED AT 4MHZ.

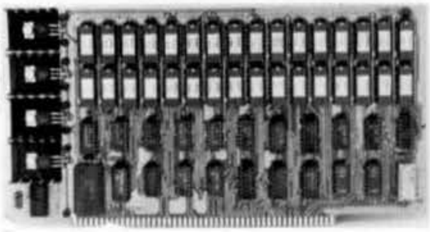
Blank PC Board w/Documentation
$29.95
Low Profile Socket Set...13.50
Support IC's (TTL & Regulators)
$9.75
Bypass CAP's (Disc & Tantalums)
$4.50

16K STATIC RAM KIT-S 100 BUSS

PRICE CUT!

$249 KIT

FOR 4MHZ ADD $25



KIT FEATURES:

1. Addressable as four separate 4K Blocks.
2. ON BOARD BANK SELE T circuitry (Cromemco Standard!). Allows up to 512K on line!
3. Uses 2114 (450NS) 4K Static Rams.
4. ON BOARD SELECTABLE WAIT STATES.
5. Double sided PC Board, with solder mask and silk screened layout. Gold plated contact fingers.
6. All address and data lines fully buffered.
7. Kit includes ALL parts and sockets.
8. PHANTOM is jumpered to PIN 67.
9. LOW POWER: under 1.5 amps TYPICAL from the +8 Volt Buss.
10. Blank PC Board can be populated as any multiple of 4K.

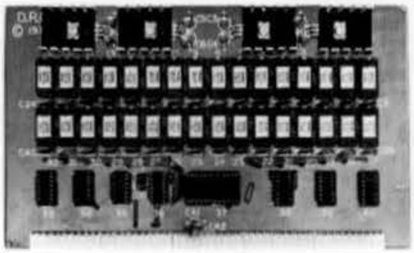
BLANK PC BOARD W/DATA-$33
LOW PROFILE SOCKET SET-$12
SUPPORT IC'S & CAPS-$19.95
ASSEMBLED & TESTED-ADD $30

OUR #1 SELLING RAM BOARD!

16K STATIC RAM SS-50 BUSS

PRICE CUT!

$239 KIT



FULLY STATIC AT DYNAMIC PRICES

FOR SWTPC 6800 BUSS!

ASSEMBLED AND TESTED - $35

KIT FEATURES:

1. Addressable on 16K Boundaries
2. Uses 2114 Static Ram
3. Runs at Full Speed
4. Double sided PC Board. Solder mask and silk screened layout. Gold fingers
5. All Parts and Sockets included
6. Low Power: Under 1.5 Amps Typical

BLANK PC BOARD—$26 COMPLETE SOCKET SET—$12
SUPPORT IC'S AND CAPS—$19.95

NEW! STEREO! NEW! S-100 SOUND COMPUTER BOARD

At last, an S-100 Board that unleashes the full power of two unbelievable General Instruments AY3-8910 NMOS computer sound IC's. Allows you under total computer control to generate an infinite number of special sound effects for games or any other program. Sounds can be called in BASIC, ASSEMBLY LANGUAGE, etc.

COMPLETE KIT!
$84.95
(WITH DATA MANUAL)

BLANK PC BOARD W/DATA $31

KIT FEATURES:

- TWO GI SOUND COMPUTER IC'S.
- FOUR PARALLEL I/O PORTS ON BOARD.
- USES ON BOARDAUDIO AMPS OR YOUR STEREO.
- ON BOARD PROTO TYPING AREA.
- ALL SOCKETS, PARTS AND HARDWARE ARE INCLUDED.
- PC BOARD IS SOLDERMASKED, SILK SCREENED, WITH GOLD CONTACTS.
- EASY, QUICK, AND FUN TO BUILD, WITH FULL INSTRUCTIONS.
- USES PROGRAMMED I/O FOR MAXIMUM SYSTEM FLEXIBILITY.

Both Basis and Assembly Language Programming examples are included.

SOFTWARE:

SCL Interpreter coming soon! Our new Sound Command Language Interpreter along with the Register Examine/Modify (REM) routines and Sound Effects Library (SEL) will be available soon in EPROM. SCL makes sound effects programming generally easier and quicker than that written in Basic or Assembly Language. An SCL users group will be formed, and the best new SCL programs submitted will be added to the Sound Effects Library in EPROM.

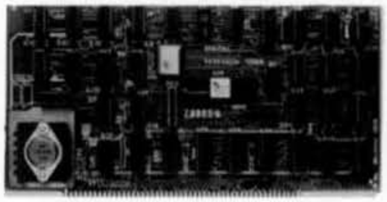
S-100 Z80 CPU CARD

Perfect for OEM's

$169.95

WIRED! NOT A KIT!

4 MHZ



ASSEMBLED AND TESTED! READY TO USE! Over 3 years of design efforts were required to produce a **TRUE** S-100 Z80 CPU at a genuinely bargain price!

BRAND NEW!

FEATURES:
★ 2 or 4 MHZ Operation. ★ Generates MWRITE, so no front panel required.
★ Jump on reset capability. ★ 8080 Signals emulated for S-100 compatability.
★ Top Quality PCB. Silk Screened, Solder Masked, Gold Plated Contact Fingers.

PRICE CUT! SALE! LOW POWER - 250NS 2114 RAM SALE! 8 FOR $49.95

4K STATIC RAMS. MAJOR BRAND. NEW PARTS.

These are the most sought after 2114's. LOW POWER and 250NS FAST.

SPECIAL SALE: $7.50 ea. or 8 FOR 49.95

NEW! G.I. COMPUTER SOUND CHIP

AY3-8910. As featured in July, 1979 BYTE! A fantastically powerful Sound & Music Generator. Perfect for use with any 8 Bit Microprocessor. Contains: 3 Tone Channels, Noise Generator, 3 Channels of Amplitude Control, 16 Bit Envelope Period Control, 2-8 Bit Parallel I/O, 3 D to A Converters, plus much more! All in one 40 Pin DIP. Supereasytointerface to the S-100 or other busses.

SPECIAL OFFER: $14.95 each Add $3 for 60 page Data Manual.

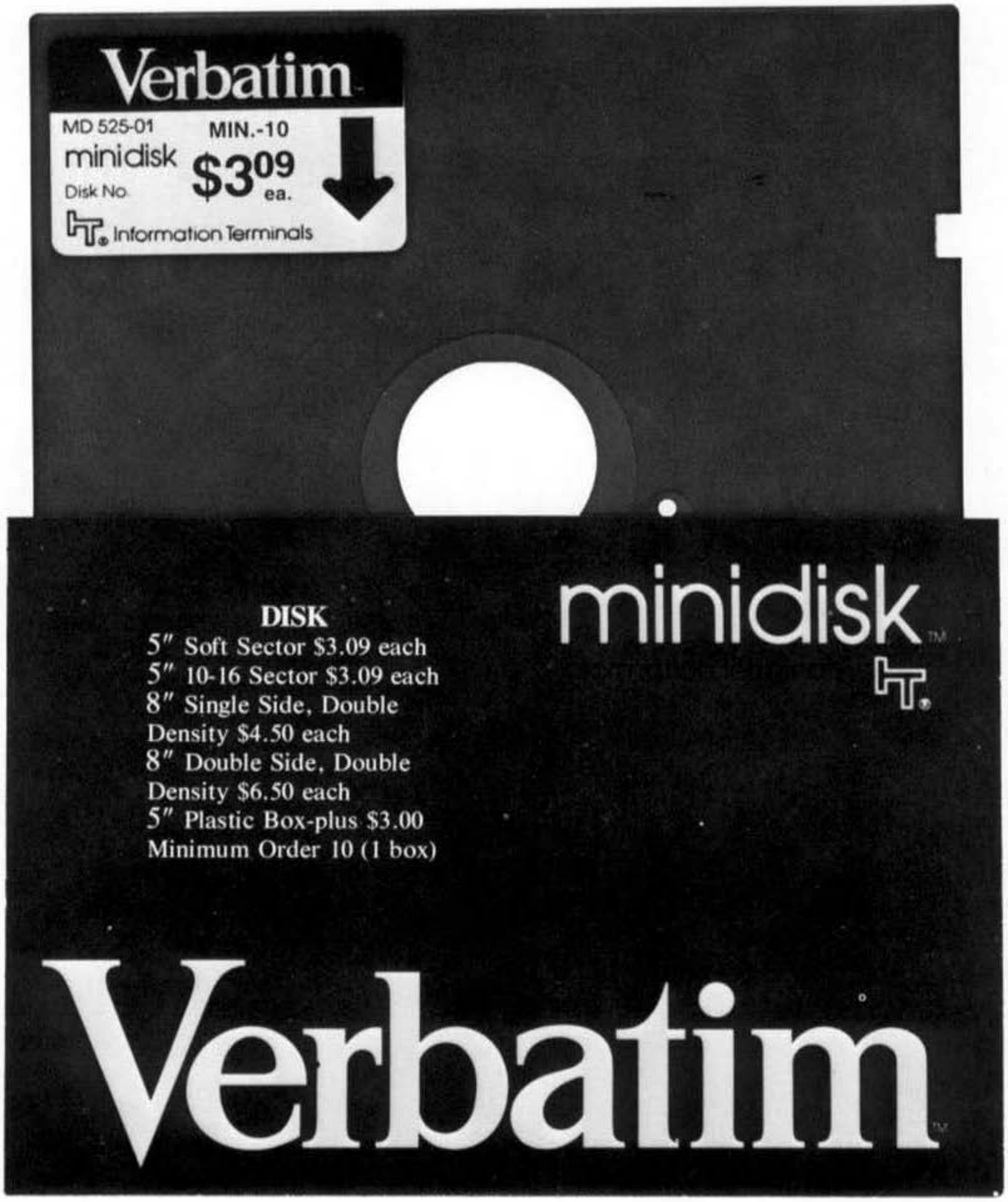
Digital Research Computers
(OF TEXAS)
P.O. BOX 401565 • GARLAND, TEXAS 75040 • (214) 494-1505

TERMS: Add $1.00 postage, we pay balance. Orders under $15 add 75¢ handling. No C.O.D. We accept Visaand MasterCharge. Tex. Res. add 5% Tax. Foreign orders (except Canada) add 20% P & H. 90 Day Money Back Guarantee on all items. Orders over $50, add 85¢ for insurance.

NOT ASSOCIATED WITH DIGITAL RESEARCH OF CALIFORNIA, THE SUPPLIERS OF CPM SOFTWARE.

5-1/4″ Minidisk — Soft or Hard Sector

Dealer and Volume Discounts Available



DISK

5″ Soft Sector $3.09 each
5″ 10-16 Sector $3.09 each
8″ Single Side, Double Density $4.50 each
8″ Double Side, Double Density $6.50 each
5″ Plastic Box-plus $3.00
Minimum Order 10 (1 box)

SAVE

DISK

SOUTH EAST MEDIA SUPPLY

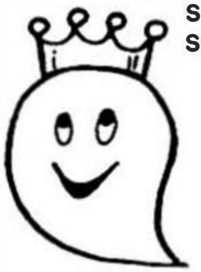
P.O. Box 794
Hixson, TN 37343
615-870-1993

H H H Enterprises
Box 493, Laurel, Md.
20810 301-953-1155

GOODIES GOODIES GOODIES

GIMIX GOODIES:

32 K static ram	$548.15
24 K static ram	$438.14
16 K static ram	$328.12
16 K ghostable	$368.16
32 K mainframe	$1594.59
80 x 24 video bd.	$398.74
80 x 24 deluxe	$458.76
8 port serial	$288.40
8 port parrallel	$198.45
8 K 2708 board	$98.34
CPU w/timers	$288.06
CPU for 6809	soon
512 x 512 graphics	soon



See GIMIX Ad on page 3

SMOKE SIGNAL GOODIES:

DCB-4 double density contrl	$499.00
SCB-69 CPU 6809	$299.00
32 K static ram	$539.00
24 K static ram	$439.00
16 K static ram	$299.00
FORTRAN 5 or 8	$149.00
DBL-SIDED 5"drv	$480.00
DBL-SIDED 8" drv	$875.00
32 K mainframe	$1595.00
32 K 09 frame	$1695.00

SWTPC GOODIES:

6809 sys kit	$495.00
Hard disk	$4395.00
16 K ROM/RAM bd.	$99.50

MOTOROLA GOODIES

BASIC09 and OS9, the best in 09 operations systems, single or multiuser, COMING SOON.

H H H ENTERPRISES

**BOX 493, Laurel, MD.
ZIP 20810
PHONE 301-953-1155**

ED SMITH'S SOFTWARE WORKS
6809 SOFTWARE TOOLS

RRMAC M6809 RELOCATABLE RECURSIVE MACROASSEMBLER. The one assembler that contains real macro capabilities (see our NOV/DEC AD). RRMAC is designed with the assembly language programmer in mind and contains many programmer convenience features. RRMAC contains a mini-editor, supports spooling or co-resident assembly, allows insert files, is romable, generates cross-references, execution times, lists target addresses of all relative references.

M69RR **$150.00**

SGEN M6809 DISASSEMBLER/SOURCE GENERATOR will produce source code (with symbolic labels) suitable for immediate re-assembly or re-editing. The output source file can be put on tape or disk. A full assembly type output listing with labels and mnemonic instructions can be printed or displayed on your terminal. Large object programs can be segmented into small source files.

M69RS **$50.00**

ANNOUNCING TWO NEW M6809 CROSS DEVELOPMENT TOOLS
CROSSBAK—A 6809 TO 6800 CROSS MACROASSEMBLER that runs on your M6809 davalopment system to produce ralocatabla M6800 object coda. Has all features of M69RR (sea above). Includes 6800 Linking Loadar.

M69CX **$200.00**

CROSSGEN—A 6800 OBJECT CODE DISASSEMBLER/SOURCE GENERATOR that runs on your M6809 development system. Has all features of M69RS (see above). An invaluable tool for converting all those 6800 object filas over to the M6809.

M69CS **$75.00**

All programs are relocatable and come complete with Linking Loader, Programmer's Guide and extensively commented assembly listing. Available on 300 Baud cassette or mini-floppy disk. For disk, specify SSB or FLEX. Source Text input/output is TSC/SSB editor/assembler compatable.

Dealer inquiries welcome. FLEX is trademark of TSC

Ed Smith's **SOFTWARE WORKS**
P.O. Box 339, Redondo Beach, CA 90277, (213) 373-3360

CT-64 CT-1024

- ★ **FAST! Average Screen Writing Speed: 19K baud**
- ★ Memory Mapped Video Adapter for your CT-64 or CT-1024.
- ★ Just plug it in and go.
- ★ The terminal works like normal until the supplied output routine is used, then the CT-64 or CT-1024 display works at processor speed.
- ★ The J.B.I. Video Board takes 1 main SS50 slot.
- ★ Video Memory can be dip switch selected to any 1K memory slot.
- ★ Combine the J.B.I. Video Board with our up and coming pseudo graphics adapter board and you put new life into the old CT-64.
- ★ The J.B.I. Video Board comes built and is jumper selectable for either the CT-64 or CT-1024.
- ★ If your terminal is CT-1024 we need to know if it is a standard 32 characters per line; or has been modified for 64 characters per line.
- ★ **SHIPPING NOW!**

The J.B.I. Video Board sells for:
$169.00 with your 21LO2'S
$179.00 with our 21LO2'S
(8-21LO2'S are required)

We have been a dealer for SWTPC since 1976.

**Johnson Micro Computer
2607 E. Charleston
Las Vegas, Nevada 89104
1-702-384-3354**

Masterchärge and Visa accepted
Dealer inquiries invited

6809!

INTRODUCING THE NEW STATE-OF-THE-ART IN MICROCOMPUTER SOFTWARE.

COMING SOON!!!

OS-9 - The most advanced microcomputer operating system available. Similar to UNIX™ (Bell Telephone Labs) with enhancements for the 6809 hardware environment. A two level firmware system. Level one supports BASIC09 and all I/O, memory management and MPU time. Level two supports all the above and inter-process communications as well as a hierarchal directory system. The addition of level two (can be done at a later date) allows full timesharing, multiprogramming real time operation for those systems supporting extended memory mapping and an interrupt-driven disk system. A minimum of 4K RAM required for level one.

A new BASIC09 language system embracing the best of BASIC, PASCAL and other popular high level languages running under OS-9 with speed and utility unmatched by other BASIC or PASCAL versions. An excellent choice for business, scientific and/or personal computer users. We feel this is the most advanced BASIC available for any microcomputer.

Call or write today for our free catalog.

MICROWARE

5835 Grand Ave. • P.O. Box 4865
Des Moines, IA 50304 • 515/279-8844

'68' MICRO JOURNAL

★ The only ALL 6800 Computer Magazine.

★ **More** 6800 material than all the others combined:

MAGAZINE COMPARISON
(2 years)
Monthly Averages
6800 Articles

KB	BYTE	CC	DOBB'S	TOTAL PAGES
7.8	6.4	2.7	2.2	19.1 ea. mo.

Average cost for all four each month: $5.88
(Based on advertised 1-year subscription price)

'68' cost per month: $1.21

That's Right! Much, Much More
for About
1/5 the Cost!

1-Year $14.50   2 Years $26.00   3 Years $36.50

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Master Charge □ — VISA □

Card #________________ Exp. Date________

For □ 1-Year   □ 2 Years   □ 3 Years

Enclosed: $____________

Name______________________________

Street______________________________

City______________State________Zip______

My Computer Is:__________________________

68 MICRO JOURNAL
3018 Hamill Road
HIXSON, TN 37343

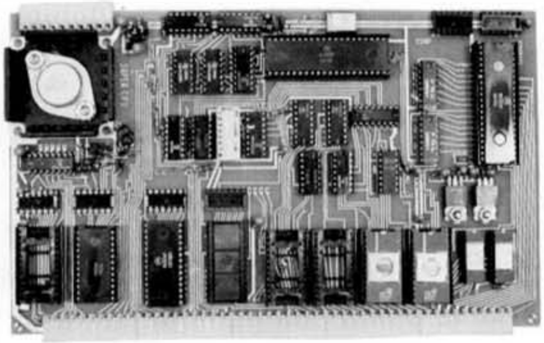
Foreign surface add $9.50 per year.
Foreign Air Mail add $29.00 per year.

Life subscription $175.00
NOTE: CANADA & MEXICO ADD $4.50 per year surface.
New subscriptions require 6-8 weeks processing.

NOW INDUSTRIAL QUALITY AT LOW COST

FROM THOMAS INSTRUMENTATION

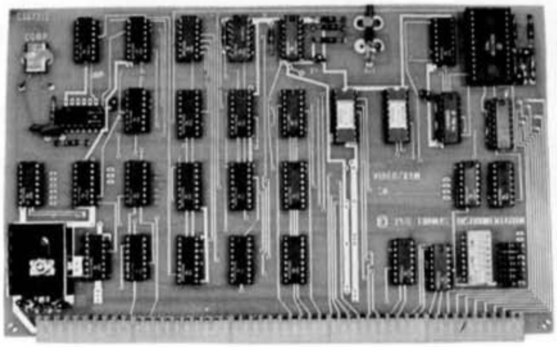
Industrial system boards are now available separately for OEM, prototyping or hobbyist applications. Shipped from stock, these are the same quality cards used in monitors and machine tool controls designed for GM, LTI, and General Electric. All cards are SS-50 buss compatible and are suitable for dedicated applications. The CPU card and the Video RAM Card may be combined on a TI backplane as a stand-alone micro — ideal for prototypes or hobbyists. ATTENTION OEM's: If you have a control data acquisition, monitoring, or other microprocessor application. Check with TI for more information about custom software design for the TI CPU or any other 6800 series system. TI also has non-SS-50 buss single board 6800 systems.



SS-50 SUPER CPU

- SS-50 or stand alone computer
- 1K of RAM at $A000 I/O on board at $A400 (Relocatable)
- 2K Monitor (Mikbug compatible) in 2708 EPROM
- 2 8 bit parallel ports with 2 control bits and power
- RS-232 ACIA port, 2nd TTL ACIA optional
- 3 16 bit counter/timers (expandable to 6 add 2nd 6840)
- 128 byte RAM at 0000 is jumper selectable
- Battery back-up for 32 bytes of RAM
- Plug back to back with Video RAM for Stand alone micro or customized smart terminal

ASSEMBLED $195.00
CARD AND DOCUMENTATION $49.00



SS-50 VIDEO RAM

- Fully synchronous operation — No jitter
- 7 by 9 Characters Programmable reverse video
- Full 128 Character ASCII set
- 1K of memory can be mapped to any 1K boundary
- Full documentation includes software (Replaces OUTEE)

ASSEMBLED $149.00
CARD, CRYSTAL and DOCUMENTATION $39.00

TI SS-50 Wire-Wrap Card	**24.00**
TI SS-50 Parallel I/O Card	**95.00**
Card only	**35.00**
GIMIX 16K Static RAM w/Soft addressing	**368.00**
GIMIX 16 RAM without Soft addressing	**298.00**
3, 4, 7 SLOT Backplanes (per slot)	**4.00**

TI cards available from stock

THOMAS INSTRUMENTATION

168-8th Street, Avalon, N.J. 08202
Phone (609) 967-4280

VISA

DEALERS FOR GIMIX SWTPC SSB

CALL FOR DEALER, OEM. AND QUANTITY PRICES

THE SOURCE

FOR UCSD PASCAL™ ON THE 6809

IMMEDIATE DELIVERY FOR SWTPC USERS, 8″ or 5¼″ DISKETTE
SMOKE SIGNAL BROADCASTING USERS, INQUIRE

CSI-1	Operating Systems, PASCAL Compiler, Screen Editor, Filer, Linker, Library, Setup, Binder, Interpreter, BIOS	$250.00
CSI-2	BASIC Compiler, YALOE, (Line-editor for hard-copy terminals), Patch, Disassembler, Calculator	$100.00
CSI-3	MACRO Assemblers for 6809, 6800 and other Microprocessors	$100.00
ALL THREE DISKS and MANUAL (SYSTEM)		$419.00

FREE! UCSD PASCAL USER'S MANUAL PLUS SWTPC IMPLEMENTATION NOTES WITH PURCHASE OF CSI-1

68000 Coming, Summer, 1980
68000 Assembler, Spring, 1980

OEM and DEALER INQUIRIES INVITED

CSI CONTROL SYSTEMS, INC.
MICROSYSTEMS DIVISION

1317 CENTRAL AVE. KANSAS CITY, KANSAS 66102

CALL TOLL-FREE (800) 255-4411
Continental U.S.A. Only
(Kansas Residents call 913/371-6136)

master charge THE INTERBANK CARD

VISA

"UCSD Pascal" is a registered trademark of The Regents of The University of California

'68' Micro Journal
3018 Hamill Rd.
Hixson, TN 37343

Second Class Postage Paid
At Chattanooga, TN
ISSN 0194-5025